Learning Bayesian Networks using Evolutionary Computation and Its Application in Classification

by

LEE Shing-yan

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

September, 2001


香港中文大學
大學
圖書館系統
23 APR 2002
UNIVERSITY
LIBRARY SYSTEM
THE CHINESE UNIVERSITY OF HONG KONG

Abstract of thesis entitled:

Learning Bayesian Networks using Evolutionary Computation and Its Application in Classification

Submitted by LEE Shing-yan
For the degree of Master of Philosophy
at The Chinese University of Hong Kong in September, 2001

In recent years, Bayesian networks are becoming popular as tools for knowledge representation. Although their applications are widespread, the learning problem, namely to learn Bayesian networks from data, is a difficult problem. In the literature, there are two different approaches to this network learning problem, namely, the dependency analysis and the score-and-searching approaches. While the two approaches handle the network learning problem differently, they both have their respective drawbacks.

Recently, there are research works that use evolutionary computation to tackle the network learning problem. Although the approach is plausible, we find that a previous formulation, which uses evolutionary programming (EP), executes slowly in practice. In this work, we, therefore, seek for ways to improve the efficiency of searching. In particular, two new strategies are proposed: (1) a hybrid learning framework (which combines the dependency analysis and the score-and-searching approaches) and (2) a novel network operator. Based on these two strategies, we come up with two different evolutionary search schemes that are named CCGA and HEP. For CCGA, the network learning problem is decomposed into sub-components using a formulation similar to cooperative coevolution discussed in the evolutionary computation literature. For HEP, it is an extension to the previous EP formulation with the new strategies incorporated.

We compare our algorithms with the original EP formulation in the experiments. Essentially, we find that both algorithms have an expected improvement in

efficiency. Because both algorithms involve a number of parameters, experiments are also conducted to investigate the effect of different parameter settings.

Besides learning Bayesian networks for the general purpose, we apply our algorithms for learning Bayesian network classifiers, including the augmented Bayesian network classifier and the multinet classifier. For performance evaluation, the classifiers are compared with other classification algorithms on different real-life data sets. Although the classifiers have also been studied in a recent research, we conjecture that the previous work employs heuristic search for classifier construction, which may yield inferior models. By using an evolutionary computation approach for learning the classifiers, we expect that our study could give a better assessment to the performance of the classifiers. Finally, we conclude our work by studying the application of Bayesian network classifiers on the direct marketing problem. Promising result is obtained which suggests that the direction is worth investigating.

內容摘要

貝氏網絡 (Bayesian networks) 是一種表達知識的工具。在現實上，它有很廣泛的應用。可是，從數據中學習貝氏網絡是一個困難的課題。從以往有關的研究中，我們可歸納出兩種不同的解難的方法：一、是利用「從屬分析」(dependency analysis)，二、是「評分—搜尋」(score-and-searching) 的方式。雖則兩種方法依循不同的理念，但是它們都有各自的問題。

最近有些研究利用「演化計算」(evolutionary computation) 來學習貝氏網絡。雖云此做法有其可取之處，但我們發現其一運用「演化編程」(evolutionary programming) 之算法在實際應用上往往執行得很慢。故此，在這項研究中，我們尋求改善效率的新方法。這包括兩項對策：一、包含了從屬分析及評分—搜尋的混合架構；二、一個新的網絡運算元。由此，我們提出了兩種不同的演化搜尋算法，名為 CCGA 與 HEP。簡單來說，CCGA 將問題細拆盼使學習能更有效率。由於此一做法與「合作性共同演化」(cooperative coevolution) 類同，故名為 CCGA。另外，HEP 則是以往的演化編程的改良版。其中最主要的改變是我們在新的的方法中包含了所建議的兩項改善。

為了評核我們新提議的算法，我們把新的算法與以往的演化編程方法作出比較。從實驗結果中，我們可以看到新的算法能在效率上有如期的改善。另外，因為我們的算法牽涉到一些參數設定，我們也從實驗裡考究了調校不同參數的影響。

除了學貝氏網絡作一般的應用，我們更將新的算法運用在學習「貝氏網絡分類器」(Bayesian network classifiers) 上。我們學習的對象包括了「增益貝氏網絡分類器」(augmented Bayesian network classifiers) 和「多重網絡分類器」(multinet classifiers)。在實驗上，我們把這些分類器和其他的分類算法作出比較。雖然過往也有相類似的研究，但我們推測前人的做法依賴於啟發式搜尋，那些所得的結果或未能盡善；簡接地，這亦影響了比較的結果。反觀之，我們則以演化計算來學習，故我們預企所得的結果會更能反映這兩種分類器的實際表現。最後，我們以一個嶄新的應用—把貝氏網絡分類器應用在直接市場推廣 (direct marketing)上，來作一結尾。

Acknowledgements

I wish to express my sincere gratitude to my supervisor Dr. Kwong-Sak Leung. Despite being occupied by his chairman duty, he is always willing to help and often gives me insightful advice. With his guidance, I have learnt to approach problems analytically, which is beneficial not only for research, but also for my future career. In addition, I would like to thank Dr. Man-Leung Wong for a series of constructive discussion. Without his help, this research work might be flawed.

I would also like to thank the members in my examination boards, especially Dr. Wai Lam and Dr. Tien-Tsin Wong. Their opinions and advice are always helpful to my research.

Thanks must be given to my colleagues. In particular, I am grateful to Tommay Tang, who sits next to me, for some friendly discussions day after day. Studying and working and with them let me learn a lot of things. Moreover, the moments that we are together, no matter playing, or dining, are often joyous.

Lastly, I want to express my heartfelt appreciation to my family. They are always an indispensable part of my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Bayesian networks, or Bayesian belief networks, are graphical representations portraying conditional independency logic. In Figure 1.1, we show an example of Bayesian networks. With each domain variable modeled as a node in a directed acyclic graph, a Bayesian network depicts the conditional independency relations among the variables. Furthermore, they also encode the joint probability distribution of the variables. Because Bayesian networks have a solid theoretical foundation, they are becoming popular as a formal knowledge representation tool that provides reasoning with uncertainty [73]. For instance, they are applied in medical diagnosis [50,56,65], information retrieval [33], and software troubleshooting [34]. Although Bayesian networks are very useful, the problem of constructing a Bayesian network is difficult.

Traditional approaches often rely on consulting experts to construct a Bayesian network about the domain. However, reliability and accuracy may be a concern because the knowledge elicitation process involves many subjective assessments and judgments. To eliminate human intervention, people are more interested in *learning* Bayesian networks from data. To be specific, the goal of the learning problem is to construct a Bayesian network which closely describes a set of past observations. In the literature, there is numerous ideas proposed that could be roughly divided into

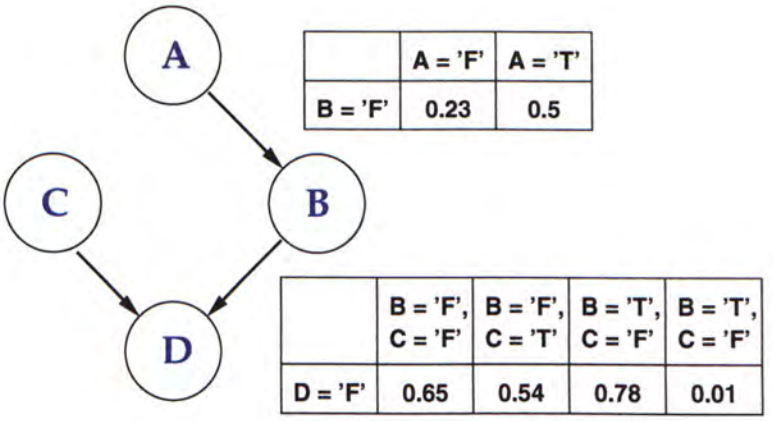


Figure 1.1: A Bayesian network example.

two categories [15]: the dependency analysis approach and the search-and-scoring approach.

Works in the first category attempt to discover dependency relations among the domain variables for constructing the network. Typically, this will involve repeated testing of conditional independence relations. Take the SGS algorithm [68,69] as an example. The algorithm begins with an undirected connected graph. In subsequent iterations, the algorithm checks whether two variables are conditionally independent between all pairs of variables by performing *conditional independence tests*. If two nodes are found to be conditionally independent, the directed edge connecting them will be removed. When no more edges can be removed, the remaining edges in the graph are oriented according to certain rules, which produces the final Bayesian network.

Works in the second category first define a metric that measures the goodness of a candidate network. With the metric, the network learning problem can be formulated as a search problem in which the goal is to find the network structure with the optimal score. For example, in the K2 algorithm [36], a Bayesian score is defined to evaluate the proximity of the probability distribution implied by a candidate network and that given by the data. Then, beginning with an empty graph, edges are added in a greedy manner so that the score is optimized at each step. Finally, the Bayesian network is constructed when no edge addition will improve the score.

Although there are two different approaches to tackle the Bayesian network learning problem, both of them suffer from some general drawbacks. For the former approach, it would require, in the worst case, an exponential number of tests to verify independency relations [69]. Moreover, some test results may be unreliable, which will greatly affect the performance of the learning algorithms. For the latter approach, the major problem is that the search space is huge. As shown in the study of Chickering et al. [17], the search problem is proved to be NP-hard when a particular metric is in use. Because the search problem is difficult, people often resort to greedy search heuristics, as demonstrated in the case of the K2 algorithm. However, the drawback is that a greedy approach may easily get stuck in local optima [32]. Although it is possible to employ a systematic and exhaustive search approach, like branch-and-bound, to obtain the global optimal solution, the involved cost will be too high in the worst case.

Recently, there are works that use evolutionary computation to tackle the search problem [47, 75]. Evolutionary computation is a general, stochastic search methodology which is often applied in large-scale optimization problems. The principal idea is borrowed from the evolutionary mechanism observed in nature, namely, "the fittest the survival." During the search, a population of candidate solutions is maintained. Then, the search space is explored by creating new solutions from the current population in each iteration. Meanwhile, only the better ones, which are evaluated according to a *fitness function*, are kept and are selected into the new population. Essentially, this amounts to let the fittest to survive while make the unfit to die. Finally, the best-so-far solution is returned.

For the network learning problem, evolutionary computation is an attractive approach as it could provide a compromise between the computational cost and the quality of the solution obtained. Because of the group searching methodology, evolutionary computation is generally expected to outperform greedy heuristics as it avoids getting stuck in a local optimal. On the other hand, when evolutionary computation is compared with exhaustive searching, it is less time-consuming and the solution returned can often be near-optimal.

1.1 Problem Statement

In this work, we study the use of evolutionary computation for Bayesian network learning. When we examine a recent work which uses evolutionary programming [75], we observe that the algorithm is unbearably slow in practice. For instance, when dealing with a data set of 37 attributes and 10,000 records, the run time will approximate 40 or more minutes. Even though the returned network is good, it is a practical concern if a program simply takes too much time to find the solution. Hence, we seek for strategies that improve the efficiency during searching.

We note that a key issue in evolutionary computation is on "exploration and exploitation." [30] On the one hand, we need to examine how do we explore the search space such that the exploration is effective. On the other hand, we need to find what is the best way to exploit our current search results. Although evolutionary computation is a general search methodology, to optimize the performance often requires the incorporation of domain-specific knowledge. In other words, if we explore and exploit in such a way that matches the particular characteristics of the problem, there will often be a reward in performance gain.

Following the same line of thought, we endeavor to devise novel learning strategies by giving a closer inspection on our problem. Based on these strategies, we shall develop efficient learning algorithms which improve over the previous approach. This is the primary goal of our work.

In addition, we could explore possible applications based on this improved learning algorithm. Because the algorithm is fast, we could test and revise some novel applications of Bayesian networks efficiently. This is the secondary goal of our work.

1.2 Contributions

We summarize our contributions as follows:

- Propose a general hybrid framework for Bayesian network learning, which improves the efficiency in searching for the optimal structure.
- Introduce a novel network operator, called *merge*. The merge operator is a recombination strategy for generating a better network from two given Bayesian

networks. It functions as an efficient genetic operator in evolutionary searching.

- Based on the two strategies, develop two network learning algorithms. In practical evaluations, the proposed algorithms provide an expected improvement in efficiency.

- Apply the network learning algorithms into learning Bayesian network classifiers. Because a previous study [26] seems to rely on heuristic search for classifier construction, the evaluation may be biased. On the contrary, we expect that our study, which uses evolutionary computation for classifier construction, can give a more fair evaluation.

- Propose a novel data mining application for Bayesian network classifiers. In particular, we evaluate the performance of a Bayesian network classifier on the direct marketing problem with comparison to the traditional approach. The study seems to reveal a worth investigating direction as suggested by the experimental result.

1.3 Thesis Organization

This thesis is organized as follows. In the next chapter, we describe the background relating to our work. This includes brief descriptions on Bayesian networks, Bayesian network classifiers and evolutionary computation.

In chapter three, we detail on two new strategies which improve the efficiency of Bayesian network learning. Next, we give a concise description on two evolutionary computation schemes that make use of the new strategies in solving the network learning problem.

In chapter four, we present the evaluations on our proposed learning algorithms. Because our algorithms involve a number of parameters, we investigate the effect of various parameter settings with discussions on some interesting patterns observed.

In chapter five, we introduce the approach that we apply our proposed algorithms in learning Bayesian network classifiers. In addition, we discuss the experimental result when we compare our classifiers with some other well-known classifiers. Finally,

we discuss a novel application by which Bayesian network classifiers are applied on a data-mining problem.

In the concluding section, we summarize our work and discuss possible future directions.

Chapter 2

Background

In this chapter, we introduce the background and previous works that are relevant to our research. In Section 2.1, we give a brief overview of Bayesian networks and their application. In order to have a self-contained discussion, we present an example and the formal definitions of Bayesian networks extracted largely from Pearl's book [57]. With the necessary background stated, we introduce the network learning problem that we endeavor to tackle. Because we apply our learning algorithms in learning Bayesian network classifiers, a brief description of the classification problem and some previous works are discussed in Section 2.2. Finally, in Section 2.3, we describe evolutionary computation in the general setting and the idea of cooperative coevolution.

2.1 Bayesian Networks

It was not until Pearl's work [57] that Bayesian networks were given a solid foundation. Basically, Bayesian networks are graphical representations which portray conditional independence logic. Before the semantics of Bayesian networks is discussed, it is best to illustrate what problems Bayesian networks target to address with an example given in [42]. Following the example, we will present the formal definitions of Bayesian networks. Next, we will describe the literature on Bayesian network learning and the applications of Bayesian networks.

2.1.1 A Simple Example [42]

Mr. Holmes has installed in his house an alarm system which could be triggered either by a burglary or an earthquake. While he is back in his office, he makes some deductions:

> "If the alarm goes off, my neighbors will probably call me by phone."
>
> "If there is an earthquake, I might notice this while listening to radio report."

By modeling each event as random variables, Mr. Holmes can represent his knowledge about the alarm system using the joint probability distribution. In other words, suppose $\{A, B, E, C, R\}$ denotes respectively the events of alarm being on, burglary, earthquake, neighbors calling and radio report, Mr. Holmes can describe the domain by specifying the entire probability distribution $P(A, B, E, C, R)$. However, such representation is unpleasant for two reasons. First, such representation is difficult to store in computers. If there are n variables (assuming they are all discrete) in the domain, each of which could take v different values, it would require n^v storage units to store the joint distribution. Second, it is rare for people to judge or reason with a joint distribution. For instance, Mr. Holmes would find the following question difficult to answer:

> "What is the probability of an earthquake without radio report and that there is no burglary and the alarm is actually on with neighbors calling?"

Although Mr. Holmes may not be able to provide the answer to the question, it will be much easier for him to estimate the probabilities of burglary happening (i.e. $P(B = 1)$), or that of the neighbors calling him when the alarm is triggered (i.e. $P(C = 1 \mid A = 1)$). In other words, people are more sensitive to the marginal probability or the conditional probability distribution of related events [57].

As an alternative, Mr. Holmes considers the well-defined notion of conditional independence in statistics literature. It comes to his mind as he notices that he tends to reason by stating the dependency relations. For example, he might have two direct conclusions:

> "Whether the alarm turns on or not depends entirely on whether there is an earthquake and whether there is a burglary."
>
> "If I know the alarm is on, it does not make a difference on my belief whether the neighbors would call even I know, in addition, that there is an earthquake."

However, Mr. Holmes guesses it will cost him hours to list all possible statements. He seeks for a concise representation in which he finds the use of graphs appealing. Not to mention that he once drew a graph to show the relations among suspects and victims, he notices that it is common for people to use graphs to illustrate dependency logic, such as a family tree.

Formalizing Mr. Holmes's idea leads to the development of Bayesian networks. With each variable modeled as a node in the graph, a Bayesian network is capable of representing conditional independence semantics. In the next section, we shall present the formal definition of Bayesian networks following largely from Pearl's book.

2.1.2 Formal Description and Notations

In probability, we have the notion of conditional independence which relates to dependency relationship. Assume that the set of domain variables U have n elements: $\{X_1, \ldots, X_n\}$; all of which are discrete and that P represents the joint probability distribution. A conditional independence relation defines a three-place relation:

Definition 1 *Let X, Y and Z be any disjoint subsets of U. We say X and Y are* ***conditionally independent*** *of Z, denoted by $I(X, Z, Y)$, if*

$$P(X = x \mid Y = y, Z = z) = P(X = x \mid Z = z) \quad \textit{whenever} \quad P(Y = y, Z = z) > 0 \tag{2.1}$$

where x, y, z are any value assignments to the sets of variables X, Y and Z respectively [57].

Intuitively, $I(X, Z, Y)$ states that having known Z, knowing the value of Y does not change our belief on the value of X and vice versa. The set of variables Z in the

above definition is called the *conditioning set.* A conditional independence relation is characterized by its order, which is simply the size of conditioning set. For example, an order-1 relation says that the two sets of variables X and Y are conditionally independent given a single variable. An order-0 relation (i.e. $I(X, \emptyset, Y)$) means that X and Y exhibit unconditional or marginal independence:

$$P(X = x \mid Y = y) = P(X = x) \quad \text{whenever} \quad P(Y = y) > 0 \tag{2.2}$$

Definition 2 *A **dependence model**, M, is the set of conditional independence assertions $I(X, Z, Y)$ [57].*

Since we can test every possible $I(X, Z, Y)$ when the underlying distribution P is given, it follows that any probability distribution defines a dependency model.

Having introduced conditional independence, the question is how can we portray the logic using a graph. Suppose we have come up with a representation scheme, a further question is how capable such model will be. In other words, we would want to know what the graphical representation describes when comparing with the dependency model implied by P. Before proceeding to answer these questions, we first define some general graph terminologies.

Definition 3 *A **directed graph** G is an ordered pair $G = (V, E)$, where V is the set of nodes and E is the set of ordered node pairs (X, Y), called directed edges or arcs. A directed edge is denoted by $X \rightarrow Y$ (or $Y \leftarrow X$). Furthermore, we say X and Y exhibit parent-and-child relationship where X is the parent and Y is the child. All parents of Y constitute the parent set of Y, which is denoted by Π_Y.*

Definition 4 *Let $G = (V, E)$ be a directed graph. A **directed path** in G is a sequence of nodes $(Y_1, \ldots, Y_m)$, $m \geq 1$, such that (Y_i, Y_{i+1}) is in E for $i = 1, \ldots, m-1$. We call the nodes that nodes precede Y_i in the sequence (i.e. $\{Y_1, \ldots, Y_{i-1}\}$ predecessors of Y_i and nodes that follow Y_i in the sequences (i.e. $\{Y_{i+1}, \ldots, Y_m\}$) descendents of Y_i. A directed path is a **cycle** if Y_1 equals Y_m.*

Definition 5 *A directed graph $G = (V, E)$ is a **directed acyclic graph** (DAG) if it contains no cycle.*

Definition 6 *Let $G = (V, E)$ be a directed graph. An* ***adjacency path*** *(or a chain) in G is a sequence of nodes $(Y_1, \ldots, Y_m)$, $m \geq 1$, such that (Y_i, Y_{i+1}) or (Y_{i+1}, Y_i), for $i = 1, \ldots, m-1$, is in E.*

Definition 7 *Let $G = (V, E)$ be a directed acyclic graph, a* ***topological ordering*** *is a total ordering of the nodes V such that $(X, Y) \in E$ implies that X appears before Y in the ordering. It is well-known that there exists a topological ordering for every directed acyclic graphs [10].*

To relate with conditional independence relations, the d-separation structure is defined:

Definition 8 *Let $G = (V, E)$ be a DAG and let X, Y be two nodes in G. Let Z be the subset of nodes of V without X and Y. An adjacency path ρ between X and Y is* ***blocked*** *by Z if one of the following is true [54]:*

- *there is a node $W \in Z$ on ρ, such that the directed edges, which determine that W is on ρ meet tail-to-tail at W.*
- *there is a node $V \in Z$ on ρ such that the directed edges which determine that W is on ρ meet head-to-tail at W.*
- *there is a node $W \in V$, for which W and none of W's descendents are in Z, on ρ such that the directed edges, which determine that W is on ρ, meet head-to-head at W.*

Definition 9 *Let $G = (V, E)$ be a DAG, and X, Y and Z be disjoint subsets of the set of nodes V. X and Y are* ***d-separated*** *by Z, denoted by $\langle X \mid Z \mid Y \rangle$ if every path between a node in X and a node in Y are blocked by Z [57].*

As each G embodies a number of d-separation configurations, it follows that G represents a dependency model if every d-separation configuration is interpreted as a conditional independence statement. Contrasting the dependency model M with G, we have the following definitions:

Definition 10 *Let M be the dependency model defined by P over the set of variables U. A graph G is a* ***dependency map*** *(or D-map) of M if there is a one-to-one correspondence between the variables in U and the nodes in V of G such that for all disjoint subsets X, Y, and Z of elements, we have [57]:*

$$I(X, Z, Y) \Longrightarrow < X \mid Z \mid Y > \tag{2.3}$$

Similarly, G is an ***independency map*** *(or I-map) of M if*

$$I(X, Z, Y) \Longleftarrow \langle X \mid Z \mid Y \rangle \tag{2.4}$$

G is said to be a ***perfect map*** *of M if it is both a D-map and an I-map of M. Equivalently, G and P are also said to be* ***faithful*** *to each other [69].*

Because it is erroneous if G contains extra conditional independence assertions than what is implied by P, I-mapness is clearly the property that is sought for. Moreover, it will be best if a graphical representation contains conditional independency assertions as much as possible. This leads to the definition of Bayesian networks.

Definition 11 *G is a* ***minimal I-map*** *of M if none of its edges can be deleted without destroying its I-mapness [57].*

Definition 12 *Give a probability distribution P on U, a DAG $G = (U, E)$ is called a* ***Bayesian network*** *of P iff G is a minimal I-map of P [57].*

Citing the corollary discussed in [57], Bayesian networks also fulfill the Markov condition in the literature [69]:

Theorem 1 *Given G and a probability distribution P on U, a necessary and sufficient condition for G to be a Bayesian network is that each variable X be conditionally independent of all its non-descendents, given its parents Π_X, and that no proper subset of Π_X satisfy this condition [57].*

Consequently, this leads to an important result:

$$P(X_1, \ldots X_n) = \prod_i P(X_i \mid \Pi_{X_i}) \tag{2.5}$$

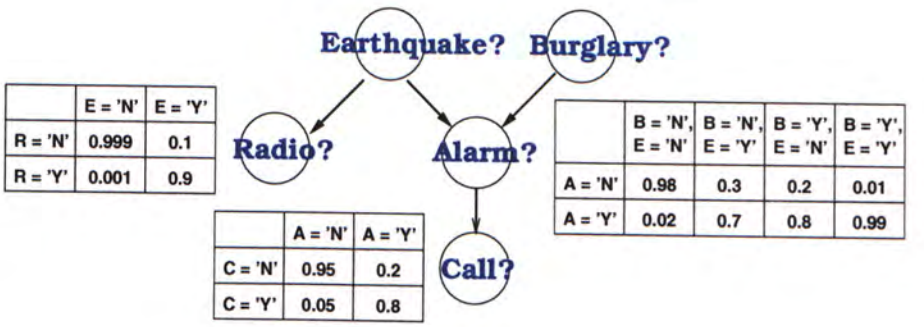


Figure 2.1: A typical Bayesian network.

Equation 2.5 indicates that the joint distribution P can be represented as a product of conditional distributions (or the marginal distribution if the parent set of a node is empty) according to the Bayesian network G of P. Owing to this result, a Bayesian network addressed the inadequacies of the joint distribution representation that we mentioned in Mr. Holmes's example, namely, the storage problem and the oddity in human's reasoning. On the one hand, since the entire distribution can be specified by n conditional distributions, this suggests an efficient representation scheme. Hence, a Bayesian network is often considered as a "compact encoding" of the joint probability distribution. On the other hand, since it is observed that people is more capable of reasoning with conditional distribution, Bayesian networks are appealing to be used as knowledge representation tools.

In Figure 2.1, we show a Bayesian network which describes Mr. Holmes's alarm system. Assume that the variables $\{A, B, E, C, R\}$ denote the occurrence of the mentioned events and each of which could take one of the two possible values: 'Y' for happened and 'N' otherwise. As can be observed, the directed edges correspond to the explicit dependence that Mr. Holmes has in his mind. In addition, the graph also contains a set of conditional independence assertions which is expressed through d-separation structure. For instance, we could readily read $I(E, A, C)$ which coincides with Mr. Holmes's reasoning. Note that besides the graphical structure, there are conditional probability tables (CPT) associated with each node which is essential for defining the joint probability distribution. (Not shown here in the figure are the marginal probabilities of nodes B and E.)

2.1.3 Learning Bayesian Network from Data

Suppose help is available from domain experts, we could construct a Bayesian network about the domain by consulting the experts. Such process is typical of building an expert system and is called *knowledge engineering* or *knowledge elicitation*. The advantage of using expert's knowledge is that the method is simple and direct. In general, the network structure as told by the experts will be a good approximation as they are knowledgeable about the domain. However, this approach has two disadvantages. First, reliability is a concern as the information obtained is largely from subjective judgements. Second, it is difficult, if not impossible, for people to estimate event probabilities precisely.

To avoid these problems, people resort to a machine learning approach, namely to learn a Bayesian network from collected data or past observations about the domain. Assume, for simplicity, that the data does not contain missing values and that there is no unobserved or hidden variable. In the literature of Bayesian network learning, we could roughly divided the works into two categories [15]: the dependency analysis and the score-and-searching approaches.

The reason that there exist two distinctively different approaches follows from the fact that Bayesian networks can be viewed differently. On the one hand, Bayesian networks are considered as depicting the underlying dependency models. In this regard, it suggests the use of dependency information for the Bayesian network construction. On the other hand, Bayesian networks are considered as encoding a joint probability distribution (equation 2.5). From this perspective, various kind of measures are devised which evaluate the quality of a given network. Consequently, the learning problem can be formulated as a search problem in which the aim is to find the best network with respect to the given measure. Although there are two different approaches for Bayesian network learning, they have, in general, respective problems and difficulties remaining to be solved.

The Dependency Analysis Approach

The first approach is called the dependency analysis approach which includes the examples in [69], [28], and [15]. Typically, it assumes the existence of a perfect map

for a given distribution P. In other words, it is assumed that there exists a Bayesian network, G, that captures all the conditional independence relations implied by P. Consequently, this suggests a general Bayesian network learning methodology: construct a network G by testing the validity of any independence assertions $I(X, Z, Y)$. In practice, we can use what is collectively called the conditional independence test (CI test) for testing. If the statement $I(X, Z, Y)$ is supported by the data, D, it follows that X should be d-separated with Y by Z in G; otherwise, X is not d-separated with Y by Z.

As a digression from the on-going discussion, we give a brief description on the CI test. One of the common approach is to use hypothesis testing procedure discussed in the statistical literature [2, 67, 69]. To begin with, the conditional independence assertion (i.e. $I(X, Z, Y)$) is modeled as the *null hypothesis*. Suppose that we use the likelihood-ratio χ^2 test, the χ^2 statistics is calculated by:

$$G^2 = -2 \sum \text{observed} * \log(\text{observed}/\text{expected}). \tag{2.6}$$

Simply put, the statistics calculates the discrepancies between the real occurrence, *observed*, and the expected count followed from the hypothesis, *expected* over every distinct events. In our case, because $I(X, Z, Y)$ implies:

$$\begin{aligned} P(X, Y, Z) &= P(X \mid Y, Z)\, P(Y, Z) \\ &= P(X \mid Y)\, P(Y, Z) \qquad \text{(by equation 2.1)} \end{aligned}$$

the statistics is computed by:

$$G^2 = -2 \sum_{x,y,z} P(x, y, z) \log \frac{P(x, y, z)}{P(y, z) P(x \mid z)}. \tag{2.7}$$

Suppose that the number of possible instantiations of the variables X, Y, and Z are respectively v_X, v_Y and v_Z, G^2 follows a χ^2 distribution with $(v_X - 1) \times (v_Y - 1) \times v_Z$ degree of freedom. Checking our computed G^2 against the distribution, we obtain the p-value, which is "the smallest level of significance for which the data leads to the rejection of the null hypothesis [5]." If the p-value is less than than a predefined *cutoff value* α, the test shows strong evidence to reject the hypothesis; otherwise, the hypothesis cannot be rejected.

Take the SGS algorithm [69] as an illustration. The algorithm begins with a completely connected undirected graph. In other words, dependence between every pair of variables is assumed. Then, CI tests between all pairs of connected nodes are conducted. When two nodes X and Y are found to be conditionally independent given Z, the undirected edge between them is removed so that $I(X, Z, Y)$ is not violated. When no more edges could be removed, the undirected edges in the graph are oriented according to some rules which conform with the conditional independence relations discovered previously. This produces the final Bayesian network.

In general, there are three problems typical to the dependency analysis approach. First, it is difficult to determine whether two nodes are dependent. Quoting from [69]: "In general, two variables X and Y may be conditionally dependent given a set Z while independent on the subset or superset of Z." In the worst case, like in SGS, every possible combinations of the conditioning set needs to be examined which would require an exponential number of tests. Second, results from CI test may not be reliable especially for high order CI tests (when the size of the conditioning set is high) [20, 69]. Hence, for algorithms that require high order CI tests, the results may be inaccurate. Third, because a network is constructed in a step by step manner, the construction algorithm may be *unstable* in the sense that an earlier mistake during construction is consequential [19, 69]. Moreover, this suggests that the order of testing the CI relations is important, which will be a concern when one pursues for the optimal performance.

The Score-and-Search Approach

The second approach is called the score-and-search approach. Recalling that a Bayesian network encodes a joint probability distribution (equation 2.5), we could derive a measure for assessing the goodness of such encoding. For instance, such measure could derived from Bayesian statistics, information theory or the Minimum Description Length (MDL) principle [63]. Though their theoretical foundations are different, some studies show that different metrics are asymptotically equivalent under certain conditions [10, 70].

Since we employ the MDL metric [45] in our work, we take it as an example for illustration. Basically, the metric is derived from information theory and incorpo-

rates the idea of the Minimum Description Length principle. The metric has two components: the network description length and the data description length. An optimal network is the one that minimizes both simultaneously.

Formally, let $U = \{N_1, \ldots, N_n\}$ be the set of discrete variables, Π_{N_i} denotes the parent set of a node N_i in the candidate network, and v_i denotes the number of possible states of the variable N_i. The network description length is given by:

$$\sum_{i=1}^{n} \left[|\Pi_{N_i}| \log_2(n) + d(v_i - 1) \prod_{N_j \in \Pi_{N_i}} v_j \right]$$

where d is a constant denoting the number of bits used to store a numerical value. Intuitively, the network description length represents the structural complexity of the network which is evaluated by the number of bits required to encode the graphical structure and to store the conditional probability table at each node.

Meanwhile, the data description length is given by:

$$\sum_{i=1}^{n} \sum_{N_i, \Pi_{N_i}} M(N_i, \Pi_{N_i}) \log_2 \frac{M(\Pi_{N_i})}{M(N_i, \Pi_{N_i})}$$

where $M(.)$ is the count of the particular instantiation in the data set. In essence, the data description length evaluates the proximity of the distributions implied by the data and the candidate network, which is a measure of the accuracy of the candidate network.

Because the MDL metric is simply the sum of the two description lengths, it puts a balance between model complexity and model accuracy. In other words, the optimal network, with regard to the metric, should be simple while accurately represents the joint distribution.

As a property common to other metrics, the MDL metric is node-decomposable and could be written as in equation 2.8. One can observe that the score is simply the summation of the independent evaluation on the parent set, Π_{N_i}, of every node N_i in the domain U.

$$\text{MDL}(G) = \sum_{N_i \in U} \text{MDL}(N_i, \Pi_{N_i}) \tag{2.8}$$

With the metric defined, the network learning problem can be formulated as a search problem. The objective is to search for the network structure which has the

optimal score. However, the problem does not become easier as the search space, that contains all possible network structures, is huge. As Chickering et al. shown, the search problem is proved to be NP-hard with the use of a particular metric [17]. Some research works, therefore, resort to greedy search heuristics [36, 45]. But the drawback is that the approach may yield sub-optimal solutions. Some others use systematic and exhaustive searching, like branch-and-bound [72] to find for the optimal solution. In the worst case, the time consumed would be considerable. Recently, there are attempts [47, 75] to use evolutionary computation to tackle the problem, which provide a compromise between the computational cost and the quality of the solution obtained.

Although numerous algorithms are proposed to address the difficulties, it is our general impression that no concluding remarks could be readily given. Suffice it to say, different approaches have their strengths and weaknesses.

2.1.4 Inference on Bayesian Networks

Since a Bayesian network encodes a joint probability distribution, it can be used to perform various kind of probabilistic inference in diagnosis or prediction. As discussed in [31], Bayesian networks are used in:

1. computing the probability of the conjunction of a set of random variables,
2. computing the most likely combination of values of variables in the network,
3. computing the piece of evidence that most influenced or will have the most influence on a given hypothesis.

Unfortunately, to perform inference on a Bayesian network is a difficult problem. Theoretically speaking, it is NP-hard to compute the exact inference result [34]. Nevertheless, there exists an efficient algorithm for a special class of networks, called the singly-connected network [31].[1] Extending the result, there are different algorithms developed that tackles the inference problem in the general cases.

[1] Also called polytree, in which there is only one adjacency path between any two nodes in the graph.

Another approach to the inference problem is to perform *approximate* inference. In particular, cases are generated from the network and the desired probability is estimated by counting. For algorithms that perform approximate inference, they are further divided into a few major categories, including instantiation-based method, random sampling, structural approximation, and loopy-belief propagation [42]. However, since the exact inference problem is NP-hard, it follows that to perform approximate inference with which the error is bounded is also NP-hard [42].

Although the inference problem remains to be difficult, different algorithms present practical methodologies or heuristics which helps to tackle the problem more efficiently. After all, since the actual performance is what is noticeable, the theoretical limitation is merely a minor concern in real world practice.

2.1.5 Applications of Bayesian Networks

With its solid foundation in probability theory, Bayesian networks provide a formal tool for reasoning with uncertainty in knowledge-based systems [73]. In recent years, Bayesian networks have widespread applications in many domains including medical diagnosis, information retrieval, map learning, language understanding, vision, heuristic search and so on [14,31]. Here, we cite a few noteworthy applications:

- In the PATHFINDER system, a Bayesian network is constructed from 105,000 cases for use in diagnosing lymph node removed from a patient examined by a pathologist [31]. The initial models consider for about 60 possible lymph node diseases. Quoted from [14], the PATHFINDER system is regarded to have achieved "expert-level" performance.

- In the Lumiére project, a Bayesian network is used in the Office Assistant in the Office'97 software application. It infers a user's goals and needs by considering the user's background [31,33].

- In the Mission Control Center of the Johnson Space Center, the Vista system uses Bayesian networks to interpret live telemetry and provides advice on the likelihood of alternative failures of the spaces shuttle's propulsion system [31].

2.2 Bayesian Network Classifiers

2.2.1 The Classification Problem in General

The classification problem is a well known problem in the machine learning community. Simply put, it ask the question: "What is the *class* (or *label*) that a given *object instance* belongs to? Indeed, classification has great practical significance as it is typical in human's reasoning. For example, in biology, we classify living beings into different species; in chemistry, we discern different elements; and in daily life, we distinguish strawberry from cranberry. Besides that the classification result is useful, the problem itself is interesting because of the simple and readily comprehensible goal it sets.

An object instance (or instance for short) is described by a collection of *feature* or *attribute* values. A feature or an attribute stands for those extracted information that is relevant to the classification task and could either be continuous (such as height) or discrete (such as sex). Assume $A_1, \ldots, A_n$ denotes the set of feature, an object instance is described by a feature vector $\{a_1, \ldots, a_n\}$ where a_i is the actual value that a feature A_i takes. As an illustration, suppose that we are classifying the type of an iris and the features $(A_1, \ldots, A_n)$ are the petal width, petal length, sepal width, and sepal length of the flower, $\{a_1, \ldots, a_n\}$ will be corresponding real measurements of the target. With the feature vector treated as input, the objective of classification is to output the correct class value $C = \{c_1, \ldots, c_k\}$ that the target object belongs to.

If an instance comes with the class information, it is called a labeled instance, otherwise, it is called an unlabeled instance. Typical in a classification problem, we are given a number of labeled instances at the first place, which may be considered as our previous observations. We call this set of data as the *training set* or the *learning set* as they provide the information necessary for constructing a *classifier*. A classifier, once constructed, functions as the device that predicts the classes of unlabeled instances. Thus, in constructing the classifier, it is important that we aim at "get(ting) the most out of the data." [74] Since every classifier presumes a particular classification model (which depends on our approach taken), learning a classifier is equivalent to finding the *best* model that fits the data.

Often, it is necessary to evaluate the performance of the classifier so as to see how good it is or to make a comparison with other classifiers [22]. Suppose that, for simplicity, the zero-or-one loss function is in use, the classifier is evaluated by the misclassification rate or the error rate on classifying unlabeled instances. In theory, the performance of the classifier is evaluated by the the *true error rate* [74], which is the error rate on classifying *every* possible instances. Since it is likely to be computationally infeasible to compute the true error rate, various performance evaluation methods are proposed which approximate the estimation of the true error rate. In the simplest case, a separate set of data, called the *testing set*, is used to evaluate the performance of the classifier. In general, when the size of the testing set is large, evaluation on the testing set could give a good approximation to the true error rate. When the data set is small, it is common to use resampling techniques like cross-validation or bootstrapping in performance evaluation. Although we shall not delve into the details, suffice it to say, resampling techniques are reminiscent to the train-and-test methodology except that the training sets and the testing sets are selected in a particular manner.

In the classification literature, there are many different approaches to tackle the problem. For example, there are approaches that use decision trees, neural networks and rule-based systems to perform classification. Also, there are works that begin with a probability point of view. We call the classifiers that falls into this category *Bayesian classifiers.*

2.2.2 Bayesian Classifiers

Theoretically, the Bayesian classifier, which follows the *Bayes decision rule*, is the best classifier in the sense that the probability of error is minimized [22]. In particular, the Bayes decision rule estimates the conditional probability of the class variable for a given instance, and returns the class which yields the greatest value. Formally put, an instance $\mathcal{I} = \{a_1, \ldots, a_n\}$ is assigned to class c_i if,

$$P(C = c_i \mid \mathcal{I}) > P(C = c_j \mid \mathcal{I}) \quad \text{for all } j \neq i. \tag{2.9}$$

By Bayes rule, the class posterior probability could be expressed as:

$$P(C \mid A_1, \ldots, A_n) = \frac{P(A_1, \ldots, A_n \mid C)P(C)}{\sum P(A_1, \ldots, A_n \mid C)P(C)}. \tag{2.10}$$

Because the denominator in equation 2.10 is the same for every $P(C \mid A_1, \ldots, A_n)$, the decision function can be rewritten as,

$$P(\mathcal{I} \mid C = c_i)P(C = c_i) \geq P(\mathcal{I} \mid C = c_j)P(C = c_j) \quad \text{for all } j \neq i. \tag{2.11}$$

Although the idea is theoretically sound, to represent $P(A_1, \ldots, A_n \mid C)P(C)$ presents a problem. If the training set is very large, it is possible to store the entire distribution in a table (assuming all attributes are discrete). However, this is impractical for two reasons. First, it is difficult to represent or store the entire distribution. Second, in real-world practice, the training set often has limited size. Thus, it is almost impossible to learn the true distribution from the training data. With such limitations, people resort to make various assumptions so as to approximate the estimation of the true distribution [74]. Bayesian network classifiers are one of the examples.

2.2.3 Bayesian Network Classifiers

Since Bayesian networks can be used to represent a joint probability distribution, we can apply them to approximate the estimation of $P(A_1, \ldots, A_n, C)$. The classifier thus built is named a Bayesian network classifier. In retrospect, the naïve Bayesian classifier could be regarded as a forerunner of the Bayesian network classifiers despite that it is proposed long before Bayesian networks become formalized. Following the naïve Bayesian classifier, there are many exciting developments in the field which attempts to improve upon the naïve Bayesian classifier. Recently, Friedman et al. [26] propose the tree augmented classifier which is regarded as the state-of-the-art Bayesian network classifier [38]. Not only that their classifier attains outstanding performance, but their comprehensive study [26] also contributes by formalizing the "Bayesian network classifier" terminology, detailing its development, making both qualitative and quantitative comparisons among various classifiers, and discussing possible extensions. Although it would be a repeat of words to speak more of

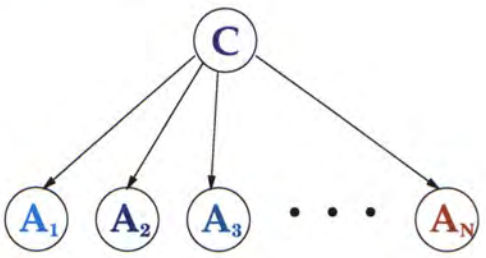


Figure 2.2: The structure of the naïve Bayesian classifiers.

Bayesian network classifiers, we still give an overview along the line of development which constitutes the necessary background to put forward our Bayesian network classifier learning algorithm.

Naïve Bayesian Classifiers

The naïve Bayesian classifier is an early attempt that follows the Bayesian classification principle. The classifier simplifies the estimation of the joint probability distribution by assuming that each attribute is conditionally independent of others given the class variable. Although the naïve Bayesian classifier exists long before Bayesian networks were formalized, its independence assumption could readily be depicted using a Bayesian network. As shown in Figure 2.2, the structure of the classifier is characterized by that every attribute node would have the class node as its parent. Formally, such independence assumption enables the likelihood probability be represented as a product of $P(A_i \mid C)$:

$$P(A_1, \ldots A_n \mid C) = \prod P(A_i \mid C) \tag{2.12}$$

Because of the naïve assumption, it is trivial to learn the classifier from data [46]. In particular, we only need to evaluate each of the n conditional probability distribution, i.e. $P(A_i \mid C)$. If all the attributes are discrete, this amounts to filling in a two-way contingency-table by counting the occurrence of each distinct instantiation (c_i, a_i) in the training set. Not only that learning is efficient, but prediction is also a trivial matter. From equation 2.12, to classify a new instance only requires multiplying the n product terms for k times.

As we would normally expect that attributes are intervened in an intricate manner, the assumption behind the naïve Bayesian classifier seems unrealistic [26]. How-

ever, in many real-life problems, the classifier often exhibits surprisingly good and robust performance [46]. Added that the classifier is simple to construct and use, it has widespread applications in many domains. Although its robustness has been inexplicable [2], the naïve Bayesian classifier initiates the development of many better classifier models. We discuss the approaches in the following text.

Two Ways to Improve the Naïve Bayesian Classifier

Following the success of the naïve Bayesian classifier, there are a number of works that attempt to make further improvements. In general, they would modify the basic assumption that is simply fallible in theory. Consider an example given by Langley et al. in [46]. Suppose that the problem domain contains three attributes A_1, A_2, A_3. By equation 2.12, the naïve Bayesian classifier calculates the posterior probability by:

$$P(A_1, A_2, A_3 \mid C) = P(A_1 \mid C)P(A_2 \mid C)P(A_3 \mid C) \tag{2.13}$$

Suppose that we include a redundant attribute A_4 such that A_4 is perfectly correlated with A_1. In particular, we could imagine that A_4 copies the value of A_1 for each instance in the training set. Hence, the conditional probability distribution of A_4 equals that of A_1, i.e. $P(A_4 \mid C) = P(A_1 \mid C)$. It turns out that the class posterior probability is given by:

$$\begin{aligned} P(A_1, A_2, A_3, A_4 \mid C) &= P(A_1 \mid C)P(A_2 \mid C)P(A_3 \mid C)P(A_4 \mid C) \\ &= P(A_1 \mid C)^2 P(A_2 \mid C)P(A_3 \mid C) \end{aligned}$$

As can be observed, the influence of A_1 is now doubled because of the existence of the redundant attribute A_4. Meanwhile, the influence due to the other attributes, A_2 and A_3 is diminished. Consequently, we say that the naïve Bayesian classifier produces a "biased prediction [46]." Although this example is imaginary and is unlikely to happen in real world problems, it nevertheless demonstrates that dependency among attributes will aggravates the performance of the naïve Bayesian classifier.

[2] Recently, there are also works [21] that attempts to give reasons for why the naïve Bayesian classifier is so good.

Among the works that try to improve over the naïve Bayesian classifier, they are roughly divide into two categories [26]:

- **Feature Selection Approach**

  Because redundant attributes may have detrimental effect on the naïve Bayes classifier. One way to circumvent the problem is to use only a subset of features for building the classifier. A noteworthy work in this category is the "Selective Bayesian Classifier" which is proposed by Langley and Sage [46]. As the authors point out, their main contribution is in extending the feature selection methodology, which has been studied in the literature, into building naïve Bayesian classifier. Nevertheless, they present a thoughtful analysis of the problem and the approach they have taken. As suggested by empirical results, they conclude that the selective Bayesian classifier retains the simplicity of the naïve Bayesian classifier yet overcome the weakness mentioned.

  Although not strictly related to the naïve Bayesian classifier, Singh and Provan also employ feature selection to learn Bayesian network classifier [65]. They call their approach "Selective Bayesian Network." In their work, they examine a number of feature selection approach and make comparison with the naïve Bayesian classifier, decision tree and a Bayesian network classifier without feature selection. Experimental result shows that the feature selection provides improvement on learning Bayesian network classifier. Furthermore, the selective Bayesian network outperforms the naïve Bayesian classifier for most of the problems and has comparable performance to decision tree. They observe that their approach excels on large dataset but may have poor performance on small data sets that have many attributes.

- **Augmented Network Approach**

  Another approach to improve upon the naïve Bayesian classifier is to remove some of the independence assumptions. Equivalently, this amounts to adding *augmenting* edges to the naïve Bayesian classifier structure, and hence the name "augmented network approach." For example, in the semi-naïve Bayesian classifier [26, 43], in contrast to assuming that attributes are conditionally independent, it assumes that different attribute *groups* are condi-

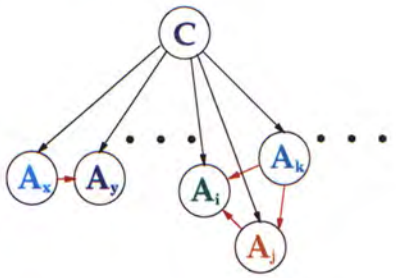


Figure 2.3: The structure of the semi-naïve Bayesian classifiers.

tionally independent while making no independence assumption on attributes within the same group. With regard to its structure, this is equivalent to having the augmenting edges to form a complete subgraph for attributes in the same group, which is illustrated in Figure 2.3.

Ezawa and Schuermann [23] proposed the "Advanced Pattern Recognition and Identification" system (APRI) which constructs an augmented network based on results from mutual information test. Since the APRI system targets on predicting uncollectible telecommunications account, they have special emphasis on minimizing the access to the large database and concern more about misclassification result. Unlike other approaches where the naïve structure is assumed, APRI chooses $C \rightarrow A_i$ depending on the mutual information between the class node C and each attribute variable A_i. Furthermore, augmented edges are added according to the class conditional mutual information (equation 2.14) between every pair of attribute variables.

Tree-Augmented Naïve Bayesian Network Classifiers

Recently, Friedman et al. propose the tree-augmented naïve Bayesian network classifier (TAN). In essence, the classifier structure contains augmented edges which forms a spanning tree. By modifying Chow and Liu's work [18], they develop an efficient learning algorithm which returns the maximum likelihood estimate of tree-augmented structures. In Figure 2.4, we show the learning algorithm from [26]. The conditional mutual information is defined by:

$$I_P(X; Y \mid Z) = \sum_{x,y,z} \log \frac{P(x, y \mid z)}{P(x \mid z)P(y \mid z)} \tag{2.14}$$

1. Compute the conditional mutual information $I_P(A_i; A_j \mid C)$ between each pair of attributes, $i \neq j$.
2. Build a complete undirected graph in which the vertices are the attributes $A_1, \ldots, A_n$. Annotate the weight of an edge connecting A_i to A_j by $I_P(A_i; A_j \mid C)$.
3. Build a maximum weighted spanning tree.
4. Transform the resulting undirected tree to a directed one by choosing a root variable and setting the direction of all edges to be outward from it.
5. Construct the classifier network by adding the class node, labeled by C, and adding an edge from C to each A_i.

Figure 2.4: The TAN learning procedure.

Intuitively, $I_P(X; Y \mid Z)$ measures the information that Y provides about X when Z is known. Let N be the size of the training set, calculating the *weights* of the edges has complexity of $O(n^2 N)$. Once the weights are obtained, the learning algorithm amounts to constructing the maximum weight spanning tree, which could be solved in $O(n^2 \log n)$. Since N is usually larger than $\log n$, the overall complexity of the learning algorithm is thus $O(n^2 N)$, which is computationally efficient.

In their work, they compare TAN with a number of existing classification algorithms, including the naïve Bayesian classifier, the selective naïve Bayesian classifier, C4.5, Chow-and-Liu multinet classifier, and other Bayesian network classifiers. On evaluating across a number of data sets, TAN shows promising performance and is evidently superior to the naïve Bayesian classifier while being competitive with C4.5 and the selective naïve Bayesian classifier.

Because of its efficient learning algorithm and its remarkable performance, TAN is regarded as the state-of-the-art Bayesian network classifier [38]. Besides that TAN earns high reputation, the comprehensive study by Friedman et al. also put forward the research on Bayesian network classifier. Since TAN, there are a number of works that follows which attempt to learn Bayesian network classifier of various kind and with different approaches [16, 38, 51, 52].

Although its performance is satisfactory, a question naturally arises: "Why should the augmented edges exist as a spanning tree?" If we consider the tree-like structure as constraints, are the constraints simplistic (like in the naïve Bayesian classifier), or the constraints stringent? In response to this, it is important to note

that the adherence to learning a tree-like network in TAN is mainly due to computational consideration [26]: while there is an efficient and theoretically sound algorithm to learn a tree-like network, there seems to be no efficient way to learn a classifier with a free structure (or more complicated than a tree). Hence, despite that an unrestricted structure is more expressive and possibly lead to a better performance, the question remains to tackle the construction problem in the first place.

2.3 Evolutionary Computation

Evolutionary computation is a general stochastic search methodology. The principal idea borrows from evolution mechanisms suggested by Charles Darwin. Since evolutionary computation is very powerful, it is often applied in solving large-scale optimization problems in different areas. For example, it is applied in data mining, image processing, pattern recognition, and signal processing [3]. Although it is difficult to evaluate formally the performance of evolutionary computation, the many successful cases suggest its versatility.

In essence, evolution computation is a group search algorithm with guidance. In Figure 2.5, we shows the typical steps during searching. A candidate solution in the search space is called a *chromosome*. A chromosome consists of a number of *genes*, which correspond to the elements constituting a solution. In the beginning, a pool of chromosomes, also called the *population*, is created randomly. As such, a number of search points are maintained. For each created individual, the fitness value, which stands for the quality of the candidate solution it encodes, is computed according to a predefined *fitness function*. In subsequent iterations, or *generations*, new chromosomes (the *offspring*) are created by genetic operators which alter the genetic composition of the parental chromosomes. Intuitively, this could be regarded as the exploration to the search space by exploiting previous search results. Then, selection comes into play where the weaker ones will vanish while stronger ones will have a higher chance to survive onto the next generation. This process is repeated until certain termination criterion is satisfied. Because only better ones will survive, it is expected that a good, or near optimal, solution can be obtained ultimately.

1. Set t, the generation count, to 0.
2. Create an initial population, Pop(t), randomly.
3. While the termination criteria is not matched,
   - Select individuals for reproduction according to fitness.
   - Apply genetic operators to to produce offsprings.
   - Evaluate the fitness of the offsprings.
   - Replace members in the Pop(t) with offspring, which gives Pop($t+1$).
   - increment t by 1
4. Return the best-so-far individual as the solution.

Figure 2.5: Procedures of Evolutionary Computation

2.3.1 Four Kinds of Evolutionary Computation

In reality, evolutionary computation is an umbrella term which includes four major implementation scheme: genetic algorithms (GA), evolutionary strategies (ES), evolutionary programming (EP) and genetic programming (GP). As we shall shortly see, each of the four differ from the others mainly on the chromosome representation and the choice of genetic operators which could bring benefits for solving a particular type of problems.

Genetic Algorithms

The basic prototype of genetic algorithms (GA) is originally introduced by Holland [37] in around 1975 which then become a stochastic search methodology. In GA, the chromosome is a bit-string of '0' and '1' which encodes the solution. Genetic operators like *crossover* and *mutation* are used to reproduce new offspring. The crossover operator takes two parent as input and produces two offspring by exchanging segments of the bit-strings of the parents. As a result, the offspring are considered to *inherit* genetic materials from the parents and possibly obtain the best of both in due course. Mutation, on the other hand, modifies the genetic composition (i.e. changes a '0' to '1' or '1' to '0') randomly so as to maintain diversity in the population. Usually, mutation is used sparingly in compared to crossover so that GA would not be degraded into a random walk.

Evolutionary Strategies

Evolutionary strategies is typically used to solve numerical optimization problem. Each chromosome is a real-valued vector which represents a solution. In each generation, the chromosomes are subject to random Gaussian perturbation that alters their numerical values. In contrast to other evolutionary algorithms, there are two main differences. First, selection in evolutionary strategies is a deterministic operator. Second, strategy parameters, which include the mutation rate and the recombination method are contained in an individual so that both the solution values and the control parameters are evolved together [3].

Evolutionary Programming

Fogel et al. first introduced evolutionary programming for learning finite state machines [3]. In essence, evolutionary programming is an evolutionary computation scheme which does not assume a fixed solution representation. Hence, a chromosome could be represented as any data structure. Different from GA or GP, EP often uses problem-specific mutation operation for creating new individuals. Moreover, tournament selection is usually employed in selection [3].

Genetic Programming

Proposed by Koza [44], genetic programming has become an active research area in recent years. GP attempts to solve complicated problems by inducing a computer program by evolution. In its original formulation [44], a program is a LISP tree. The nodes of the tree are the *function set* and the *terminal set* which describes the program. Similar to GA, GP also uses crossover operator and mutation operator for producing new solutions. Since the target is a tree structure, the crossover operator now exchanges subtrees between the parents and that the mutation operator changes a node randomly. In contrast to other evolutionary computation methods where a solution to an optimization problem is returned, GP is more powerful as it can evolve complicated logic expressed in a program. Recently, the expressiveness of GP is enriched by the introduction of a better representation and more complicated operations. It is the expectation of Koza that computers could at-

tain human-competitive intelligence for solving difficult problems by using genetic programming.

2.3.2 Cooperative Coevolution

Although we have presented the general framework of evolutionary computation in Figure 2.5, there are many variations possible. Sometimes, the ideas originate from the observation in nature which is absent from the original framework. For instance, the notion of exons and introns [3], sexuality, and niching have been introduced in the literature. In general, they target to overcome the inadequacies of the original framework. Among the many variations, we will introduce the idea of cooperative coevolution which is used in our work.

Coevolution is the evolution of different species in the same environment, where the interactions among them affect the genetic composition of each other. There are two types of coevolution: competitive and cooperative. Often, the type of coevolution we observed in nature belongs to competitive coevolution. In competitive environment, the genetic compositions of a species evolve so as to enhance its competitiveness in the environment. For instance, the "arm race" between two species is a case of competitive coevolution. For cooperative coevolution, the natural selection pressure will favor individuals that could form good collaboration with other species.

Based on the work of Potter and DeJong [59–61], cooperative coevolution represents a problem breakdown methodology. A problem instance is divided into a number of subcomponents that corresponds to different species. The analogy is that once species (i.e. solutions to subcomponents) could cooperate among themselves, the collaboration (i.e. the assembled solution) will be a good solution.

In each species population, evolutionary search is conducted separately. During fitness evaluation, an individual is assigned a fitness value so that cooperation is promoted. To achieve this, a collaborative structure $\mathcal{S}$ is first assembled from representatives of each species population. Note that $\mathcal{S}$ now is a complete solution to the original problem. When an individual is subject to fitness evaluation, it replaces its representatives in $\mathcal{S}$ and forms $\mathcal{S}'$. As such, the individual is assigned with the

1. Set t, the generation count, to 0.
2. For each species k,
   - Create an initial population, $\text{Pop}_k(t)$, randomly.
3. For each species k,
   - Evaluate the fitness of individuals in $\text{Pop}_k(t)$.
4. While the termination criteria is not matched,
   - For each species k,
     - Evaluate the fitness of individuals in $\text{Pop}_k(t)$.
     - Select individuals for reproduction according to fitness.
     - Apply genetic operators to produce offsprings.
     - Evaluate fitness of the offsprings.
     - Replace members in $\text{Pop}_k(t)$ with offspring, which gives the new population $\text{Pop}_k(t+1)$.

Figure 2.6: Cooperative Coevolution Algorithm

fitness of $\mathcal{S}'$ which reflects, to a certain degree, how good it cooperates with other species. Figure 2.6 shows the cooperative coevolution algorithm.

By using cooperative coevolution, a hard and complex problem could be handled in a systematic and efficient manner. For example, cooperative coevolution is applied in learning neural networks [60] and in learning of sequential decision rules [61].

Chapter 3

Bayesian Network Learning Algorithms

In this chapter, we focus on the use of evolutionary computation for Bayesian network learning. As have discussed before, with a metric defined, the learning problem becomes a typical search problem. In particular, we have to search for the network structure which scores the best according to the metric. Since the search space, which consists of all possible directed acyclic graph structures, is huge, it is justified to use evolutionary computation for the search problem.

In the literature, there are two previous pieces of work which use evolutionary computation for Bayesian network learning. The first one, proposed by Larrañaga et al. , uses genetic algorithm (GA) while the second one, proposed by Wong et al. , uses evolutionary programming (EP). In the work by Wong et al., the two approaches are compared and it is found that the EP formulation performs better than the GA one in general.

In practice, we found that the EP approach often has slow convergence rate which results in a long execution time. To improve the efficiency, we propose two novel strategies for the network learning problem. First, we introduce a general hybrid framework which is a combination of the dependency analysis and the score-and-search approaches. Second, by noting the characteristics of the score evaluation, we devise a novel network operator which generates a better network structure out of two parent networks.

Based on these strategies, we come up with two different learning algorithms. The first one, which we is called CCGA, uses the idea of cooperative coevolution for searching. Essentially, CCGA decomposes the network search problem into subcomponents, which is fast but may be vulnerable for difficult problem instances. The second one, which is called HEP, is a modification of the EP approach which incorporates the two new strategies and some minor improvements. In comparing the two approaches with the original EP approach, it is found that our new formulations have significant improvement in efficiency.

This chapter is organized as follows. Section 3.1 describes the two previous approaches in brief. Section 3.2.1 and 3.2.2 present the hybrid approach and the new operator respectively. In Section 3.3, we describe the CCGA approach. In Section 3.4, we describe the HEP approach. Finally, we summarize our contributions in Section 3.5

3.1 Related Work

In this section, we review the previous work which uses evolutionary computation as the search strategy in Bayesian network learning. Mainly, there are two pieces of work: the first one uses genetic algorithm (GA) while the later one uses evolutionary programming (EP). In the following discussions, we shall use the notation $U = \{N_1, \ldots, N_n\}$ to denote the set of variables (or nodes) in the problem domain and use Π_{N_i} to denote the parent set of the node N_i.

3.1.1 Using GA

Larrañaga et al. [47] proposed using genetic algorithms for searching the optimal Bayesian network structure. In their work, the network structure (composed of n nodes) is represented by an $n \times n$ "connectivity matrix" C which is, in effect, the transpose of the adjacency matrix. Each element C_{ij} in the matrix is defined as:

$$C_{ij} = \begin{cases} 1, & \text{if node j is a parent of node i} \\ 0, & \text{otherwise.} \end{cases}$$

With this representation, the $i-$th row in the matrix encodes the parent set of node N_i (i.e. Π_{N_i}). An illustration is given in Figure 3.1.

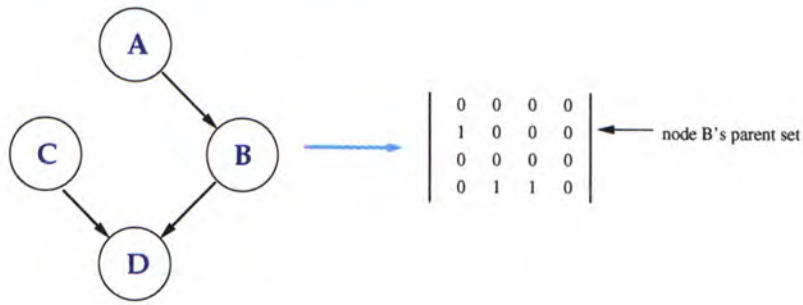


Figure 3.1: Matrix representation of a DAG.

By flattening the matrix, the bit-string representation is obtained:

$$C_{11}C_{21}C_{31}\ldots C_{n1}C_{21}C_{22}\ldots C_{nn}$$

With such representation, traditional GA (with one-point crossover and mutation) is performed to search for the optimal solution. For the fitness function, they adopt the Bayesian score (referred as the BD score in [17]) which is used in the K2 algorithm [36]. Note that since the genetic operators could generate illegal offspring structures (i.e. networks that are not DAG), cycle repairing is needed after an offspring is produced.[1]

Because it is rare to have a densely connected network in real world problems, they have imposed a limit on the number of parents a node could have in their implementation.[2] Even though such restriction greatly reduces the possible search space, the problem is still NP-hard as suggested by the work of Höffgen [47].

They have conducted a number of experiments to test the GA approach with different implementations under different parameter settings. Based on the results, several recommendations regarding the choice of implementation and parameters are made.

[1]In a later work, they consider the case when a node ordering is given. With a node ordering, the genetic operators become closed operators which means no repairing is required.

[2]The same limit is imposed in MDLEP and all of our algorithms.

3.1.2 Using EP

Recently, Wong et al. [75] used evolutionary programming (EP) to tackle the structural search problem. Since they use the Minimum Description Length metric [45] to evaluate fitness, they call their approach MDLEP.

EP is different from GA mainly in the format of solution representation and the genetic operators used. Unlike the restricted use of string in GA, EP does not have any restriction in solution representation. An individual in MDLEP is simply the connectivity matrix of the network. Furthermore, there is no crossover operation in EP, and the only operation is mutation. An outline of the MDLEP algorithm is given in Figure 3.2.

---

1. Set t, the generation count, to 0.
2. Create an initial population, Pop(t) of m random DAGs (m is the population size).
3. Each DAG in the population is evaluated using the MDL metric.
4. While t is less than the maximum number of generations,
   - Each DAG in Pop(t) produces one offspring by performing a number of mutation operations. If there is cycle, a randomly picked edge in the cycle is removed.
   - The DAGs in Pop(t) and all new offspring are stored in the intermediate population Pop'(t). The size of Pop'(t) is $2 \times m$.
   - Conduct a number of pairwise competitions over all DAGs in Pop'(t). For each G_i in the population, q other individuals are selected. Then the fitness of G_i and the q individuals are compared. The score of G_i is the number of individuals (out of q) that has lower fitness than G_i.
   - Select the m highest score individuals from Pop'(t) with ties broken randomly. The individuals are stored in Pop($t+1$).
   - increment t by 1.
5. Return the individual that has the lowest MDL metric in any generation of a run as the output of the algorithm.

Figure 3.2: The MDLEP algorithm.

In essence, MDLEP uses four mutation operators which includes simple mutation, reversion mutation, move mutation and knowledge-guided mutation. The simple mutation operator picks a random edge and either adds the edge to (when it is already present) or removes it from (when it is absent in network) the network. The reverse mutation operator picks an edge from the network in random and re-

verses its direction. The move mutation operator modifies the parent set of a node by replacing one of the parents with a non-parent. The knowledge-guided mutation operator is similar to simple mutation except that an edge is selected with certain guidance. Briefly, each edge, $X \rightarrow Y$, is weighted by evaluating the MDL score of node Y having only X as its parent. With such weights as the guidance, the mutation operator removes an edge with the heaviest weight or adds an edge with the lightest weight among a pool of candidate edges.

In their experiments, they test their algorithm with data sets generated from two benchmark networks, ALARM and PRINTD. In comparing with the GA approach using the MDL metric, they found that MDLEP performs better in many aspects. In general, those networks generated from MDLEP produce smaller structural difference (in compared to the original network), and smaller average MDL score. In addition, MDLEP is also faster as it requires fewer generations to converge and generates less invalid structures.

3.1.3 Criticism of the Previous Approaches

As reported in Wong et al.'s work, the EP formulation seems to have a clear advantage over the GA one. To account for this, we conjecture that the performance gain is largely due to the different choice of genetic operators in producing the offspring, which, in effect, influences the exploration of search space. On the one hand, the success of EP readily suggests that sheer mutation, which corresponds to adding or removing an edge or the mix of the two, is good enough for generating new search points. On the other hand, the crossover operation, which plays an important role in GA, seems to be ineffective. The reason is not difficult to understand as the one-point crossover, in our case, effectively, recombines two graph arbitrarily. In most cases, this could result in an invalid structure. In this regard, such recombination would seem to be insignificant and offspring do not properly *inherit*, which is supposedly the merit of the crossover operator. Although the intention to exchange information among the population is good, the traditional one-point crossover fails to achieve the purpose.

Despite that the EP approach performs better, we note that it often requires

a long execution time. For instance, for learning a network with 37 nodes from a given data set of 10,000 cases, MDLEP needs about an hour to find the solution, [3] which is intolerable for any practical use. At a closer look, we find that a major reason that MDLEP converges slowly is because there are much more worse offspring (comparing an offspring with its parent) produced than better offspring on average. From our experience, if the population size is 50, we would have, on average, less than five better offspring produced in each generation (see Section 4.2.2 for details). Effectively, this implies that most of the mutation operation only results in creating inferior network structures. Hence, we conjecture that MDLEP is not efficient enough in finding good solutions.

3.2 Two New Strategies

3.2.1 A Hybrid Framework

Having discussed that there are two drastically different approaches for Bayesian network learning, it is straight-forward for people to develop a hybrid of the two approaches. In the literature, there are several attempts which show different realizations of the idea. For instance, the CB algorithm [66] employs a score-and-search approach (i.e. K2) which takes as input a node ordering given by the network constructed by a dependency analysis approach. In another scenario, the BENEDICT algorithm [1] uses a metric definition which reflects the discrepancy between the independency assertions implied by the network and the data. In our work, we propose a new hybrid framework which targets to improve the efficiency of Bayesian network learning. In essence, information from dependency analysis is exploited in the score-and-search process so that the searching will be more efficient.

Typical in dependency analysis approach, it is assumed that a perfect map (c.f. Section 2.1) exists for a given data set D. In particular, we assume the existence of a Bayesian network G which depicts every conditional independency relation as implied by D (i.e. $I(X, Z, Y)$) through d-separation. To check the validity of a conditional independency assertion $I(X, Z, Y)$ of any given two nodes X, Y and

[3] We use 5,000 generations as the termination criterion.

a conditioning set Z, CI tests are often used. In a straightforward sense, if the assertion $I(X, Z, Y)$ is valid, X and Y cannot be connected because this will violate that X and Y are being d-separated (see Definition 9). In other words, neither the edge $X \rightarrow Y$ nor the edge $X \leftarrow Y$ will present in the resultant network G. In the SGS algorithm, the same rationale is used in constructing a Bayesian network in the initial steps where X—Y is removed from an undirected connected graph for each verified assertion $I(X, Z, Y)$.

With such observation, we formulate a general hybrid framework for Bayesian network learning which consists of two phases. In the first phase, we conduct low-order CI tests so that we know which edge could be removed. Note that we limit ourselves to use only low-order CI tests as their test results are more reliable than higher order CI tests while the time complexity is bounded. In the second phase, we use a score-and-search approach together with the knowledge obtained previously. In particular, we refine the search space by excluding networks that contain the edges $X \rightarrow Y$ and $X \leftarrow Y$ for every verified assertions $I(X, Z, Y)$. Consequently, the search space is reduced which would imply a speedup for the search process because we would not waste time in adding (or removing) potentially *wrong* edges.

The proposed framework is general in the sense that it does not necessitate a particular conditional independence test for the CI test phase, or a particular search method in the search phase. For instance, we could use conditional mutual information test coupled with branch-and-bound searching for learning Bayesian networks. In any case, however, we expect an enhancement in efficiency in compared to a pure score-and-searching approach.

3.2.2 A New Operator

During the course of searching, we encounter a number of candidate network structures. Although they are not the desired solution, it is very likely that they contain some good *partial structures*. Hence, we ask the question, "Given two networks, how could we, by recombining the two, produce a better one?" In examining the problem closely, it follows that we should determine what *module* is to be exchanged. A module could be anything in a network: edges, nodes, some clustering of the nodes

or even a subgraph. If a module is poorly defined, recombination of the networks may not be useful at all. So that the effort is not futile, we claim that a module needs to carry the same (or greater) significance after recombination.

As an example, let us consider the one-point crossover operator which is widely used in genetic algorithm. Basically, the crossover operator is a recombination strategy which creates offspring from parents. In the case of Bayesian network learning (discussed in Section 3.1.1), the operator creates an offspring by recombining different parts of a randomly split graph from the parents. Unavoidably, this would result in generating illegal structures (i.e. graph that contains cycle). It follows that even if an exchanged part appears in the optimal network, such part will lose its significance after recombination. Consequently, the operator is not as eff. $I(X, Z, Y)$) throug

conditional independency assertion $I(X, Z, Y)$ of any given two nodes X, Y and

[3]We use 5,000 generations as the termination criterion.

a conditioning set Z, CI tests are often used. In a straightforward sense, if the assertion $I(X, Z, Y)$ is valid, X and Y cannot be connected because this will violate that X and Y are being d-separated (see Definition 9). In other words, neither the edge $X \rightarrow Y$ nor the edge $X \leftarrow Y$ will present in the resultant network G. In the SGS algorithm, the same rationale is used in constructing a Bayesian network in the initial steps where X—Y is removed from an undirected connected graph for each verified assertion $I(X, Z, Y)$.

With such observation, we formulate a general hybrid framework for Bayesian network learning which consists of two phases. In the first phase, we conduct low-order CI tests so that we know which edge could be removed. Note that we limit ourselves to use only low-order CI tests as their test results are more reliable than higher order CI tests while the time complexity is bounded. In the second phase, we use a score-and-search approach together with the knowledge obtained previously. In particular, we refine the search space by excluding networks that contain the edges $X \rightarrow Y$ and $X \leftarrow Y$ for every verified assertions $I(X, Z, Y)$. Consequently, the search space is reduced which would imply a speedup for the search process because we would not waste time in adding (or removing) potentially *wrong* edges.

The proposed framework is general in the sense that it does not necessitate a particular conditional independence test for the CI test phase, or a particular search method in the search phase. For instance, we could use conditional mutual information test coupled with branch-and-bound searching for learning Bayesian networks. In any case, however, we expect an enhancement in efficiency in compared to a pure score-and-searching approach.

3.2.2 A New Operator

During the course of searching, we encounter a number of candidate network structures. Although they are not the desired solution, it is very likely that they contain some good *partial structures*. Hence, we ask the question, "Given two networks, how could we, by recombining the two, produce a better one?" In examining the problem closely, it follows that we should determine what *module* is to be exchanged. A module could be anything in a network: edges, nodes, some clustering of the nodes

or even a subgraph. If a module is poorly defined, recombination of the networks may not be useful at all. So that the effort is not futile, we claim that a module needs to carry the same (or greater) significance after recombination.

As an example, let us consider the one-point crossover operator which is widely used in genetic algorithm. Basically, the crossover operator is a recombination strategy which creates offspring from parents. In the case of Bayesian network learning (discussed in Section 3.1.1), the operator creates an offspring by recombining different parts of a randomly split graph from the parents. Unavoidably, this would result in generating illegal structures (i.e. graph that contains cycle). It follows that even if an exchanged part appears in the optimal network, such part will lose its significance after recombination. Consequently, the operator is not as effective as expected and is outweighed by mutation operations. In this regard, it seems that "a random part of a graph" gives a poor definition of a module.

So, how should we define a module?

Recalling that the MDL score of a network is decomposable (i.e. evaluated on the parent set of each node), it readily suggests that we could view a network as an aggregation of the parent sets (of the nodes). Thus, we regard the parent sets as modules to be exchanged. Formally, let M_i^x denotes the MDL score of the parent set $\Pi_{N_i}^x$ of node $N_i \in U$ in the network G_x. (Hence, the score of G_x equals $\sum M_i^x$.) We restate our objective as follows:

> Suppose we are given two input networks G_a and G_b, can we create a better network, G_c, by selecting $\Pi_{N_i}^c = \{\Pi_{N_i}^a, \Pi_{N_i}^b\}$ for $i = 1 \ldots n$, so that,
> (1) there is no cycle in G_c and
> (2) the the score of G_c, $\sum M_i^c$, is less than $\sum M_i^a$ or $\sum M_i^b$?

If the domain has n variables, the optimal solution (i.e. $\sum M_i^c$ is lowest while no repairing is necessary) can be found by considering exhaustively all 2^n combinations. However, this would be computationally expensive because we have to check for cycle formation for the every combinations. Hence, instead of pursuing an optimal recombination, we propose a heuristic approach, which is fast, for solving the problem. We call our proposed approach the *merge* operator.

The Heuristics

In short, the merge operator attempts to form a better network by modifying a network G_a with *certain parent sets* from another network G_b. Specifically, the operator finds a subset of nodes, $W \subset U$, with which we replace $\Pi^a_{N_j}$ with $\Pi^b_{N_j}$ in G_a for every $N_j \in W$. Moreover, provided that such modification does not create any cycle, the new network will have a score no worse than G_a. We present the pseudo-code for the merge operator as in Figure 3.3.

Procedure merge(G_a, G_b)

1. Find $\delta_i = M^a_i - M^b_i$ for every node $N_i \in U$.
2. Produce a node ordering L by sorting δ_i in descending order.
3. While there are still nodes in L left unconsidered,
   - Get the next node, N_i, from L which is unconsidered.
   - Invoke the procedure `findSubset(`N_i`, `W'`)` which returns W' on completion.
   - Sum δ_j for every node $N_j \in W'$.
   - If the sum is positive, mark every node $N_j \in W'$ in L as considered. Insert W' to W.
4. Replace $\Pi^a_{N_j}$ with $\Pi^b_{N_j}$ for every node $N_j \in W$.

Figure 3.3: Pseudo-code for merge()

For the two input networks G_a and G_b, the merge operator first produces a node ordering by sorting $\delta_i = M^a_i - M^b_i$ in descending order. Since positive δ_i means that $\Pi^b_{N_i}$ is better than $\Pi^a_{N_i}$, we follow the ordering in considering the parent set replacement (i.e. replace $\Pi^a_{N_i}$ with $\Pi^b_{N_i}$). Beginning with the first node, N_i, in the ordering, we invoke the procedure `findSubset(`N_i`, `W'`)` (shown in Figure 3.4) for finding a subset of nodes W' such that by replacing $\Pi^a_{N_j}$ with $\Pi^b_{N_j}$ for every $N_j \in W'$ in G_a, the resultant graph is still a DAG.

Initially, for an input node N_i, `findSubset()` checks whether by replacing $\Pi^a_{N_i}$ by $\Pi^b_{N_i}$ would create a cycle in G_a. The parent set replacement involves two steps: (1) set $\Pi^a_{N_i} = \phi$ and (2) set $\Pi^a_{N_i} = \Pi^b_{N_i}$. However, since step one does not affect the test result, it suffices to detect for cycle formation by step two. In other words, we only examine whether the adding the edges (i.e. $N_k \rightarrow N_i$ for every $N_k \in \Pi^b_{N_i}$) to

Procedure findSubset(N_i, W')

1. If N_i is already in W', return immediately.
2. Insert N_i to W'.
3. For every parent N_k of N_i in G_b,
   - For every node N_j in W',
     - check if there is a directed path going from N_j to N_k in G_a, i.e. $N_j \rightarrow \cdots \rightarrow N_k$.
     - If a directed path exists, invoke `findSubset(`N_k`, `W'`)`. (recursion)

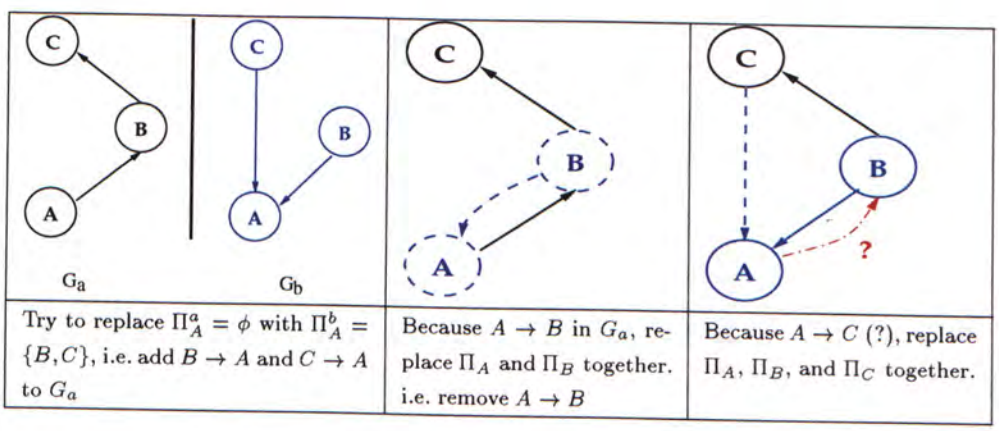
Figure 3.4: Pseudo-code for findSubset()

G_a would produce a cycle. Hence, we check whether there is a directed path going in the reverse direction ($N_i \rightarrow \cdots \rightarrow N_k$) in G_a for each edge addition. If adding all $N_i \leftarrow N_k$ do not create a cycle, we could safely replace $\Pi^a_{N_i}$ with $\Pi^b_{N_i}$. But if it is found that adding $N_k \rightarrow N_i$ will create a cycle, we consider the replacing the parent sets of node N_i and its parent N_k together by invoking `findSubset()` recursively.

In the midst of recursion, suppose that N_i still denotes the input node. However, now W' will contain the sets of nodes that we consider for parent set replacement. If N_i not already contained in W', it is added to W'. Otherwise, the procedure returns immediately which guarantees that the procedure would eventually terminate. Similar to the initial invocation, we check whether by replacing $\Pi^a_{N_i}$ by $\Pi^b_{N_i}$ would create a cycle. This time, however, the checking is more complicated because we have to cater for the simultaneous replacements of the parent sets of the nodes in W'. In particular, suppose V denotes all successors of N_i (in G_b) that are in W' (i.e. $N_i \rightarrow \cdots \rightarrow N_j$ for every node $N_j \in V$), adding the edges (i.e. $N_k \rightarrow N_i$ for every $N_k \in \Pi^b_{N_i}$) would imply that N_k is a predecessor of every N_j after the replacement (i.e. $N_k \rightarrow N_i \cdots \rightarrow N_j$). Hence, it is necessary to check the existence of a directed path from every N_j (successors of N_i, including N_i) to every N_k (every parents of N_i).

In the actual implementation, we make two approximation to realize the idea. First, we use W' in place of V. But since W' contains V, the checking is still safe. However, if we detect the existence of a directed path going from N_j to N_k, this may not signify a cycle formation as N_j may not really be in V. Second, the current

implementation checks the the existence of the reversed path in G_a. The correct approach should check the statement in an intermediate G'_a which is formed by the parent set replacements of every nodes in W'. Alternatively, it could be viewed as though we ignore the "step one" that we mentioned previously. As a result, this will, again, give a more stringent test of which the existence of a directed path may not imply cycle formation in the resultant network. We give an illustration in Figure 3.5.



Try to replace $\Pi_A^a = \phi$ with $\Pi_A^b = \{B,C\}$, i.e. add $B \to A$ and $C \to A$ to G_a	Because $A \to B$ in G_a, replace Π_A and Π_B together. i.e. remove $A \to B$	Because $A \to C$ (?), replace Π_A, Π_B, and Π_C together.

Figure 3.5: An illustration of an erroneous conclusion.

Consequently, the procedure returns the set W' with which if we place $\Pi_{N_j}^a$ with $\Pi_{N_j}^b$ for every node $N_j \in W'$ in G_a, the resultant graph is still acyclic. After we have obtained W', we calculate the sum $\sum_{N_j \in W'} \delta_j$. If the sum is positive, we insert W' into W and remove W' from the ordering. We carry on the same procedures with the next node in the ordering and end until every nodes are considered. Finally, we replace $\Pi_{N_j}^a$ with $\Pi_{N_j}^b$ in G_a for every $N_j \in W$. Note that although by replacing Π_{N_j} for every $N_j \in W'$ will not create cycle, the simultaneous replacement of different W' may create a cycle. This could easily be illustrated by an imaginary case as in Figure 3.6. In practice, however, we find that such case seldom occurs although cycle checking will still be necessary.

An Analysis of the Operator

As we have claimed before, the merge operator is a heuristic approach for selecting W. In other words, it is not guaranteed that the choice is optimal. Our primary

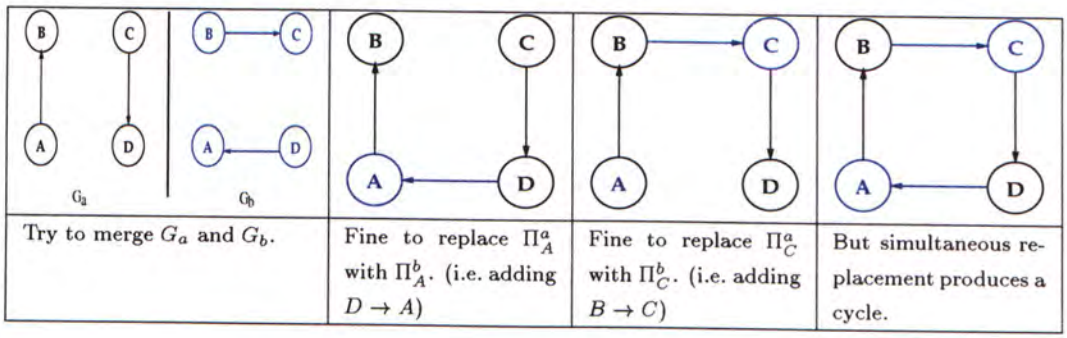


Figure 3.6: An illustration of cycle formation in an imaginary case.

concern is the computational cost of the operator. In analyzing the procedures, we obtain the worst case complexity of invoking `findSubset()` which is $O(n^2)$ and occurs when `findSubset()` follows every edges in the densest network [4]. Inside the `findSubset()` procedure, the most expensive operation is the checking of the existence of a directed path. But if we are given directed path connectivity information about G_a, it reduces to a trivial table lookup operation. Hence, our claim that the merge operator is a fast heuristic is justified. In practice, we find that the merge operator is very efficient, and could produce a better network in most cases.

Essentially, the merge operator brings us two main benefits. First, it provides a way to combine previous search efforts. Second, since the score of the composite network can be readily calculated, we need not invoke the time-consuming procedure for evaluating MDL score. This offers an economical way to create new structures.

In the next section, we shall describe two approaches that employ the mentioned new strategies to two very different extents.

3.3 CCGA

Since the MDL score of a network is decomposable, we could readily break down the network search problem into smaller components. Specifically, rather than searching for an optimal network, we attempt to search for the "optimal parent set" for each node. By "optimal parent set", we do not refer to the one with the best score, but

[4]Since each DAG conform to a topological ordering, the densely connected graph will have $n(n-1)/2$ edges.

the parent set which appears in the optimal network. With such decomposition, we introduce the use of cooperative coevolution in searching, by which we expect better efficiency could be attained.

If the sub-problems are independent among themselves, cooperative coevolution is effective in finding the optimal solution to a seemingly complex problem. Unfortunately, our problem does not perfectly fit the requirement. On the one hand, because we can evaluate the MDL score of a parent set for a given node, we can deal with each sub-problem independent of others. On the other hand, however, it is necessary that the ultimate solution is an acyclic graph. Obviously, whether a network is acyclic depends on how the network is connected. Thus, it is necessary to have the global picture in mind when solving the sub-problems, which we believe is what the "cooperation" is meant for.

To handle such intricacy, we propose a number of modifications on the original cooperative coevolution framework. In addition, the hybrid approach is employed so that our search space is reduced. Since our algorithms essentially uses GA together with cooperative coevolution, we call our new algorithm CCGA.

3.3.1 The Algorithm

With the hybrid framework, our algorithm is divided into two phases: the CI test phase and the cooperative coevolution search phase. In the first phase, CI tests are conducted to obtain a draft of the network which serves to reduce the model search space in the next phase. In the second phase, cooperative coevolution is used to search for a good network structure within the refined search space. Figure 3.7 provides an outline of the algorithm.

Before going into details, it is necessary to clarify our problem statement and the assumptions. For the given data set, we assume that there is no missing value or hidden variable and we assume no prior knowledge about the structures. Similar to previous approaches that use GA and EP, we assume that a node could not have more than k parents. With no particular preference, we evaluate the quality of a candidate network using the MDL metric. [5]

[5] Although MDL is used, other metrics, like Bayesian score could also be used

- For each node, initialize the possible parent set to contain every other nodes.
- CI Test Phase
  - Perform CI test (up to order-1) between all pairs of nodes.
  - If the nodes are found to be conditionally independent, remove each other from their possible parent set.
- Cooperative Coevolution Search Phase
  1. Set t, the generation count, to 0.
  2. For each species population,
     - Randomly initialize each chromosome in accordance with the possible parent set.
     - Evaluate the fitness of each chromosome.
  3. Compose $\mathcal{S}$ by combining the best chromosome from each species population.
  4. Pass the constraints from $\mathcal{S}$ to each species population.
  5. While t is less than the maximum number of generations,
     - Inside each species population,
       * Temporarily change the possible parent set with the given constraints.
       * Perform selection.
       * Evolve new offspring using crossover and mutation
       * Evaluate the fitness of each chromosome.
     - Update $\mathcal{S}$.
     - Produce a node ordering from $\mathcal{S}$ and pass the constraints to each species population.
     - Update the best-so-far structure.

Figure 3.7: The CCGA algorithm.

3.3.2 CI Test Phase

Initially, we let the possible parent set of each node to contain all other nodes. Using the hybrid approach discussed before, we attempt to reduce the size of the parent set of each node by discovering low order CI relations. For example, if the node X is found to be conditionally independent of node Y in a test, X will be removed from Y's parent set and vice versa. Alternatively, it could be view as though the edges $X \leftarrow Y$ and $X \rightarrow Y$ are both excluded for further consideration. Since higher-order CI tests may be unreliable, we only use order-0 and order-1 tests for discerning possible conditional independence relations.

In our implementation, we use the likelihood-ratio χ^2 test for testing. For a given assertion $I(X, Z, Y)$, a p-value is returned from the test (see Section 2.1.3

for details). If the p-value is greater than a predefined cutoff value α, the assertion cannot be rejected and we assume $I(X,\ Z,\ Y)$ to be valid.

Suppose that there are n variables. For a given pair of variables, we need to, in the worst case, conduct the order-0 test (i.e. $I(X, \Phi, Y)$) and all order-1 tests (i.e. test $I(X, Z, Y)$ for every $Z \in U \setminus \{X, Y\}$). Hence, the overall complexity of the CI test phase is bounded by $O(n^3)$ test.

Although CI tests are very useful, incorrect results could be detrimental. In particular, if a crucial edge (which appears in the optimal network) is excluded due to our findings in CI test phase, it is impossible to obtain the optimal solution in the search phase which follows. In our implementation, we choose a *moderate* α value with which we hope to lessen the reliance on the test results. We provide a study on the issue in Section 4.3.1.

3.3.3 Cooperative Coevolution Search Phase

As mentioned before, cooperative coevolution is a kind of problem breakdown. In our case, we divide the the network learning problem of n variables into n sub-problems, the objective of which is to find the "optimal" parent set for each node. Alternatively, it could be viewed as though we divide the matrix representation into rows as is illustrated in Figure 3.8. Consequently, each candidate solution to the

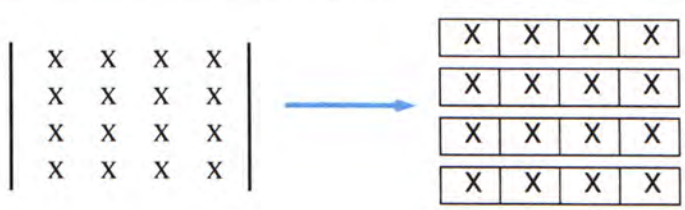


Figure 3.8: Decomposition of the matrix representation into rows.

sub-problems is represented as a bit-string, which readily suggests the use of genetic algorithm for solving the problems. Following the convention, we call the search population of each sub-problem *a species population.* Suppose there are n variables, there will be n different species populations. Inside each population, we use simple GA to search for the optimal solution.

Although such problem breakdown seems plausible, it is required that the composite network must be acyclic. Obviously, illegal solution could be avoided only if each species population has the knowledge of others and work cooperatively to prevent cycle formation. To realize this idea, we propose an approach which makes use of the topological ordering of a graph as a guidance. In the following sections, we shall discuss our algorithm in detail. Nevertheless, it must be stated that there are many possibilities to tackle the acyclic restriction. For instance, we could devise a punitive scheme such that we penalize each illegal structure produced by reducing the score significantly. Here, we just propose a viable alternative.

A Feedback Mechanism

Noting that every legal network (i.e. an acyclic graph) conforms to a topological ordering, it follows that we could use an ordering as constraints for each species population so as to avoid cycle formation. In particular, the possible parent set of a node, which defines the search space of the corresponding species population, could only consist of nodes preceding it in the given ordering. Consequently, a composite of the solutions from the populations will be acyclic (as it conforms with the given ordering). Alternatively, it could be viewed as if we are to search the optimal network for a given ordering.

Using this idea, we propose a feedback mechanism. Essentially, we use the node ordering implied by the collaborative structure, $\mathcal{S}$, to produce constraints for each species population such that each candidate solution will conform with the ordering. Next, we update $\mathcal{S}$ with results from the species populations and start another cycle. We picture this idea in Figure 3.9.

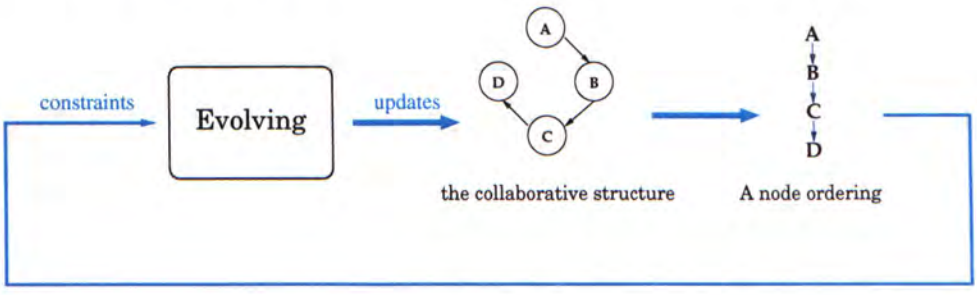


Figure 3.9: The feedback mechanism.

Nevertheless, there is a flaw in the feedback mechanism, namely, $\mathcal{S}$ will conform to the same ordering for whatever update is made. Eventually, this will drive the search process to return the optimal network for an initial ordering, which is fine only if the ordering is optimal.

To correct this problem, we, therefore, use $\mathcal{S}$ only to *approximate* an ordering. In particular, every directed edge $X \rightarrow Y$ in $\mathcal{S}$ is associated with certain degree of belief (i.e. by throwing a dice) that it also appears in the optimal structure. If our degree of belief is less than a fixed threshold, the *belief factor*, it suggests doubt on the correctness of the edge. Hence, we allow the presence of Y in X's possible parent set, which is otherwise forbidden. As a result, a new $\mathcal{S}$ could exhibit an ordering which is different from the original one. However, the drawback is that we should watch out for possible cycle formation.

Initialization

At the beginning, the chromosomes in the species population are randomly initialized. Nevertheless, the parent set limit and the result from CI test are taken into consideration. After the population is initialized, we assemble $\mathcal{S}$ using the best individual from each population. Note that, in this way, $\mathcal{S}$ will probably contain cycle(s). Next, for each node N_i in the network, we create a new network $\mathcal{S}'$ which copies $\mathcal{S}$ except that N_i is a root node in $\mathcal{S}'$ (i.e. its parent set is empty). Then, the network $\mathcal{S}'$ is repaired for cycles. Using $\mathcal{S}'$ as reference, we produce constraints on the species population of node N_i such that we assure each candidate solution, when substituted to $\mathcal{S}'$, will create a network that is still acyclic.

Searching inside the Species Populations

For every species population, the search space is equivalent to the possible parent set of the corresponding node. As mentioned before, the possible parent set of a node is subject to different changes during the course of searching. Consequently, the corresponding search process of the node faces both *permanent* and *temporary* constraints. For the permanent constraints, we refer to the reduction of the possible parent set due to the result from CI test. For the temporary constraints, we refer to

the changes due to the aforementioned ordering implied by $\mathcal{S}$. As a result of these constraints, the length of the bit-string and the mapping (i.e. which bit corresponds to which parent) are varying. [6]

With the varying bit-string representation, simple GA with crossover and mutation are used to create a new population. However, instead of using one-point or two-point crossover, we devise a modified crossover operator. Since we have a parent size limit, large part of the bit-string is empty. As an illustration, suppose that the possible parent set size is 30, and that k equals five, the bit-string will largely be empty and contains a few '1's. As a result, one-point or two-point crossover will be very likely to exchange empty segments which creates nothing new.

To circumvent this problem, we use an approach similar to uniform crossover [4]. For the two bit-strings that take part in crossover, we create two different masks for each of them. As an illustration, Figure 3.10 shows the mask defined and the bit-strings. Here, suppose that the size of the possible parent set is six and hence the bit-strings are six-bit long. A mask of equal length is created for each parent. At position where the original string is '0', the mask is undefined. At position where the original string is '1', the mask has a value of either '1' or '0'. It has '1's only at the second bit, the fourth bit and the sixth bit. Hence, the mask value is undefined for the first bit, the third bit and the fifth bit. If the mask value is '0', the corresponding

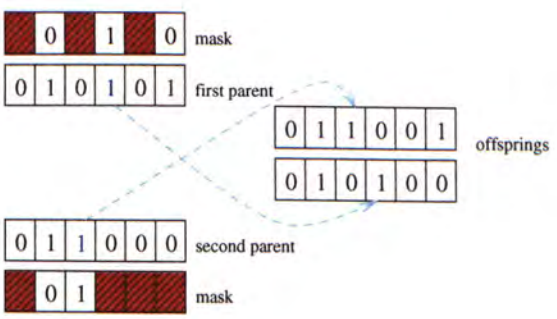


Figure 3.10: The modified crossover operator.

bit is copied to its offspring. Otherwise, if the mask value is '1', the corresponding

[6] In the special case when the size of possible parent set of a node is too small, it would clearly be unwise to search for the optimal combination using genetic operators. Hence, for such cases, we will simply list all the possibilities with the entire population and we inhibit any genetic operation to be performed on the population.

bit is copied to the offspring of the other party. Referring to Figure 3.10, the upper mask have the value '1' at fourth bit position, the corresponding bit is copied to the other offspring. The action is indicated by the arrow in the figure.

With such crossover operator, we hope to have a uniform mixing of two given parent sets. From our experience, this modified crossover operator indeed improves over a two-point crossover operator.

Update of $\mathcal{S}$

After the new population is created and evaluated at each species population, individual that has a better score than its correspondent in $\mathcal{S}$ will be used to update $\mathcal{S}$. Assume a better parent set is found for node N_i, it will replace the parent set of node N_i in $\mathcal{S}$. However, due to the relaxation of the ordering constraint, such substitution may create an edge conflict such that $X \leftarrow Y$ and $X \rightarrow Y$ coexist in $\mathcal{S}$. [7] Hence, in updating $\mathcal{S}$, we differentiate those edge additions that will create a conflict from those will not. For those that are not in conflict, they are incorporated directly to $\mathcal{S}$.

To determine which edge ($X \leftarrow Y$ and $X \rightarrow Y$) should be kept is equivalent to determine the proper orientation of the undirected edge X—Y. For this, we use the approach: when we know more about the rest of the network, we would know how an edge should be oriented. Simply put, with the remaining part of the network known and fixed, we try all possibilities of the un-oriented part. Finally, the best configuration is incorporated into $\mathcal{S}$.

For the set of conflicting edges, *related* ones are first group together. By related edges, we refer to edges that share a common set of nodes, W. Next, different configurations of orientating the group of related edges are tried and evaluated. Since the MDL metric is node-decomposable, it suffices to evaluate the total MDL score of W. Note that while the present configuration is tried, the parent set of each node contains every other known parents. Since each conflicting edge has two orientation possibilities, there are 2^m configurations to try for a group of m

[7] To be specific, there are two cases of conflict. First, $X \rightarrow Y$ and $Y \rightarrow X$ are both found on the list of update. Second, $X \rightarrow Y$ is found while there is no update for node X and $Y \rightarrow X$ is contained in $\mathcal{S}$.

edges (m is usually small). Finally, the configuration that gives the best score is used to update $\mathcal{S}$. This process is then repeated for the other groups. Since the best configuration of a group may contain cycle, the resultant $\mathcal{S}$ is repaired for cycle unavoidably. Although it is possible to check for cycle when trying different orientation possibilities, we suggest to avoid it as the involved cost will be great.

Update of the Best-so-far Structure

With such framework, the score of $\mathcal{S}$ will not converge in most cases. In other words, $\mathcal{S}$ is not the currently best solution, rather, it is more like a locally optimal search point (under certain constraints). Therefore, to obtain the best during the course of searching, we have maintained a best-so-far structure separately. In each generation, we try combining the currently best-so-far structure with $\mathcal{S}$ using the merge operator. If a better structure is created, the new structure will become the best-so-far structure.

Even though $\mathcal{S}$'s score is not converging, it possibly contains some good partial structures which is the major motivation for us to use the merge operator. By accumulating the essence into the best-so-far structure, it is expected that a good solution is obtained ultimately.

3.4 HEP

Our second algorithm is a natural extension of MDLEP. Essentially, we incorporate three major changes to MDLEP so as to improve its efficiency. First, we use the hybrid framework for reducing the search space. Unlike CCGA, we employ a novel realization which circumvents the requirement of selecting a proper α. Second, we use the new operator, merge, in place of most of the mutation operations which we hope to produce more better individuals. Third, noting that cycle repairing often takes a significant portion of the running time in MDLEP, we avoid the formation of cycle altogether when producing new individuals. With all these modifications, we call our new approach HEP (hybrid EP). An outline of the algorithm is given in Figure 3.11. In the following sections, we will discuss HEP in detail.

CI test Phase
For every pair of nodes (X, Y),

- Perform order-0 and all order-1 CI tests.
- Store the highest p-value in the matrix Pv.

Evolutionary Programming Search Phase

1. Set t, the generation count, to 0.
2. For each G_i in the population Pop(t),
   - initialize the alpha value randomly.
   - refine the search space by checking the alpha value against the Pv matrix.
   - Inside the reduced search space, create a DAG randomly.
3. Each DAG in the population is evaluated using the MDL metric.
4. While t is less than the maximum number of generations,
   - select $m/2$ individuals from Pop(t), the rest are marked "NS" (not selected).
   - For each of the selected ones,
     - merge with a random pick from the dumped half in Pop$'(t-1)$.
     - If merge does not produce a new structure, mark the individual with "NS".
     - otherwise, regard the new structure as an offspring.
   - For each individual marked "NS",
     - produces an offspring by cloning.
     - alters the alpha value of the offspring randomly.
     - refine the search space by checking the alpha value against the Pv matrix.
     - changes the structure by performing a number of mutation operations. Note that cycle formation is prohibited.
   - The DAGs in Pop(t) and all new offspring are stored in the intermediate population Pop$'(t)$. The size of Pop$'(t)$ is 2*m.
   - Conduct a number of pairwise competitions over all DAGs in Pop$'(t)$. For each G_i in the population, q other individuals are selected. The fitness of G_i is compared against the q individuals. The score of G_i is the number of individuals (out of q) that are worse than G_i.
   - Select the m highest score individuals from Pop$'(t)$ with ties broken randomly. The individuals are stored in Pop$(t+1)$.
   - increment t by 1.
5. Return the individual that has the lowest MDL metric in any generation of a run as the output of the algorithm.

Figure 3.11: The HEP algorithm.

3.4.1 A Novel Realization of the Hybrid Framework

Similar to CCGA, we also use the hybrid framework in HEP but with a different realization. If we take the approach of CCGA, we would use the test results as *global* constraints such that we forbid every candidate network to have those potentially wrong edges. However, in so doing, we take the risk of assuming our choice of α is valid. In general, smaller values of α implies more constraints (less likely to reject an hypothesis) and results in a more limited search space. If we make an improper choice, we will have, in the worst extremes, either that all edges are pruned away or that every edges are retained.

As a better alternative, we let every individual in the EP population to have a different α. Thus, each individual in the EP population has, besides the network structure, a cutoff value α which is also subjected to change. With different α, individuals will have a different, nevertheless reduced, search space. In other words, we make the results from CI test *local* to every individual.

To realize this idea, we record, in the CI test phase, the largest p-value returned by the CI tests for every possible conditioning set, Z (restricted to order-0 and all order-1 tests) and store the value into a matrix, Pv. In the search phase, for a given individual G_i in the population with associated cutoff value α_i, we forbid adding an edge $X \leftarrow Y$ if Pv_{XY} is greater than α_i (i.e. $I(X, Z, Y)$ is assumed to be valid). In the current implementation, the value of α_i is changed in a simple way. At the beginning, we randomly initialize the value of each α_i in the population. In subsequent generations, an offspring will inherit the cutoff value from its parent with possible addition and subtraction of a fixed increment, Δ_α. Hence, if a parent has the cutoff value equals α_p, its offspring will have a value of $\alpha_p + \Delta_\alpha$ or $\alpha_p - \Delta_\alpha$ if the value is ever subjected to change. Since the cutoff value is bounded between 0 and 1, both subtraction to $\alpha_p = 0$ and addition to $\alpha_p = 1$ will leave the cutoff value remain unchanged and the offspring will inherit the same cutoff value as its parent.

As a consequence of the selection pressure, individuals having an improper choice of α will eventually be killed. On the one hand, if the value of α of an individual is too small which disallows the addition of some *important* edges, the individual will have a greater chance of being killed. On the other hand, if the value of α is too

large, offspring produced from the individual will be less likely to add the *right* edges (many *wrong* alternatives). Consequently, this also leads to the eventual extinction of the individual.

3.4.2 Merging in HEP

One of the reason that MDLEP runs slowly is that there are more worse offspring produced than better offspring. To improve this situation, we favor a heavy use of merging than mutation because it is both computationally efficient and effective. In our current implementation, we select half of the population randomly for merging and only keep the results if better networks are produced. For the rest of the population and the ones which fail to give a better network after merging, we will produce offspring by mutation as usual.

For the networks that are selected for merging, they are merged with dumped networks from the last generation. In such a way, we considerably reuse our previous search efforts, which are thrown away otherwise.

3.4.3 Prevention of Cycle Formation

As we have pointed out before, MDLEP consumes much time in repairing networks. To solve this problem, we prevent cycle formation in every candidate network by maintaining the *connectivity matrix* which contains the count of directed paths between all pairs of nodes. As an illustration, if we have $X \rightarrow \cdots \rightarrow Y$ in a network, we will forbid adding the edge $X \leftarrow Y$ to the network.

Let $\mathcal{C}$ denotes the connectivity matrix and $\mathcal{C}_{XY}$ be the count of directed paths going from X to Y. $\mathcal{C}$ is updated when an edge is either added or removed. Using the convention that a node X is an immediate successor of itself, we have $\mathcal{C}_{XX} = 1$ for every node $X \in U$. The procedure for updating the matrix is shown in Figure 3.12. As can be seen, the overhead for the update has a complexity of $O(n^2)$.

Procedure UpdateConnectivityMatrix(X, Y, adding)

FOR each ancestor of X, X',
  FOR each successor of Y, Y',
    IF *adding*,
      $C_{X'Y'} = C_{X'Y'} + C_{X'X} \times C_{YY'}$
    ELSE,
      $C_{X'Y'} = C_{X'Y'} - C_{X'X} \times C_{YY'}$
    END
  END
END

Figure 3.12: Pseudo-code for UpdateConnectivityMatrix().

3.5 Summary

In this chapter, we begin by discussing the previous approaches of Bayesian network learning using evolutionary computation. Although the EP approach, called MDLEP, outperforms a GA approach, we find that it is inefficient for several reasons. Next, we propose two new strategies for the learning problem. The first one is a hybrid approach for Bayesian network learning while the second one is the merge operator which is a recombination strategy. Using these two new strategies, we present two new learning approaches. The first one, called CCGA, makes use of the idea of cooperative coevolution in decomposing the problem. However, since such decomposition also brings problem, we suggest a number of significant modifications to the basic framework of cooperative coevolution. Our second algorithm is an extension of MDLEP. To improve its efficiency, we incorporate the hybrid framework and the merge operator into the implementation. In the next chapter, we shall give a performance evaluation on the two algorithms.

Chapter 4

Evaluation of Proposed Learning Algorithms

In this chapter, we study of the performance of the two proposed Bayesian network learning algorithms, CCGA and HEP. In Section 4.1, we describe our experiment methodology. In Section 4.2, we report the experimental results on comparing the two learning algorithms with MDLEP. In Section 4.3, we give an analysis of the performance of CCGA. In Section 4.4, we give a similar analysis on HEP. Finally, we summarize our findings in Section 4.5.

4.1 Experimental Methodology

A common practice to access the performance of a Bayesian network learning algorithm is to test the algorithm on data sets that are generated from known network structures using probabilistic logic sampling [35]. Here, we follow the practice and test our algorithms on seven different data sets. All of the data sets are generated from well-known benchmark Bayesian networks which include the ALARM network, the ASIA network and the PRINTD network. Table 4.1 gives a summary of the data sets that we used in our experiments.

ALARM-1000, ALARM-2000, ALARM-5000, ALARM-1000 and ALARM-O are generated from the ALARM network which has the structure shown in Figure 4.1 [1].

[1] We reference the variable names as provided in [69]

Data set	Original Network	Size	MDL Score of Original Network	Source
ALARM-1000	ALARM	1,000	18,533.5	MDLEP
ALARM-2000	ALARM	2,000	34,287.9	MDLEP
ALARM-5000	ALARM	5,000	81,223.4	MDLEP
ALARM-10000	ALARM	10,000	15,8497.0	MDLEP
ALARM-O	ALARM	10,000	138,455.0	PowerConstructor
ASIA-1000	ASIA	1,000	3,416.9	PowerConstructor
PRINTD-5000	PRINTD	5,000	106,541.6	MDLEP

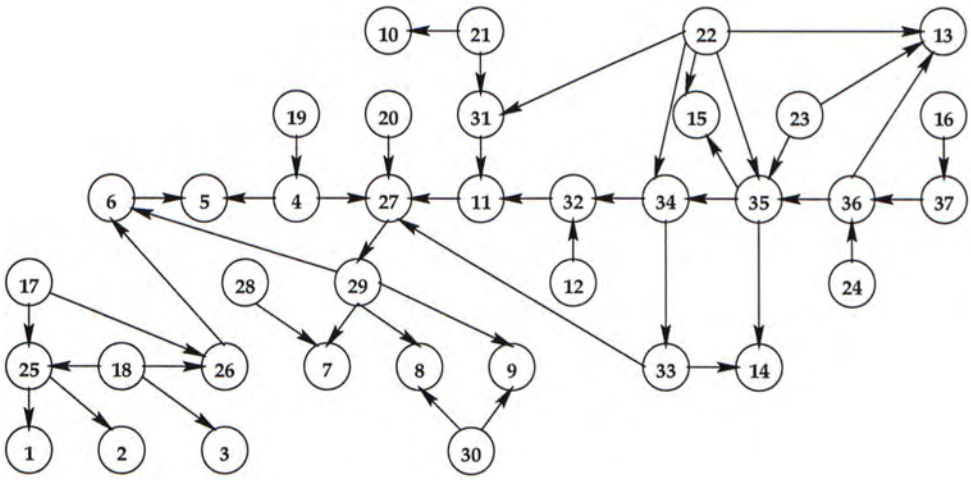
Table 4.1: Data sets used in the experiments.

Originally, the ALARM network is used in the medical domain for potential anesthesia diagnosis in the operating room [6]. Because the network, with 37 nodes and 46 directed edges, has a complex structure, it is widely used for evaluating the performance of a learning algorithm. Examples include the K2 algorithm [36], the CB algorithm [66], the BENEDICT algorithm [1], and MDLEP [75]. The ALARM data sets that we used in our experiment are obtained from two different sources. One of the data set (ALARM-O) is obtained from the PowerConstructor software package [15] which contains 10,000 cases. For the rest(i.e. ALARM-1000, ALARM-2000, ALARM-5000, and ALARM-10000), they are obtained from the authors of MDLEP, which have been used for evaluating their algorithm. The four data sets are of different sizes and contain 1,000, 2,000, 5,000 and 10,000 cases.

Also from the authors of MDLEP, we obtain a data set with 5,000 cases, which is generated from the PRINTD network. The PRINTD network is primarily constructed for troubleshooting printer problems in the Windows™ operating system [34]. The structure of the network is shown in Figure 4.2. It has 26 nodes and 26 edges.

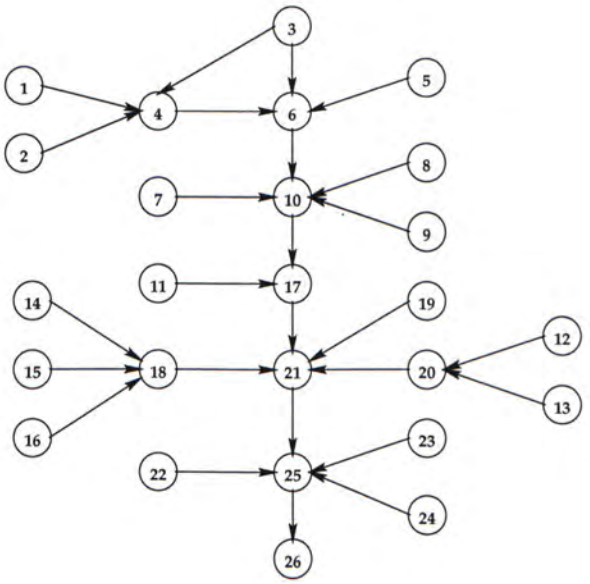
One of the data set is generated from the ASIA network. As shown in Figure 4.3, the ASIA network, is a relatively simple structure that contains eight nodes and eight edges. The network is also known as the "chest-clinic" network which describes a "fictitious medical example whether a patient has tuberculosis, lung cancer or bronchitis, related to their X-ray, dyspnea, visit-to-Asia and smoking status [48,55]." The data set we used contains 1,000 cases.

In our experiment, we compare the performance of our algorithms with MDLEP. All algorithms (including the implementation of MDLEP which is obtained from



1. central venous pressure
2. pulmonary capillary wedge pressure
3. history of left ventricular failure
4. total peripheral resistance
5. blood pressure
6. cardiac output
7. heart rate obtained from blood pressure
8. heart rate obtained from electrocardiogram
9. heart rate obtained from oximeter
10. pulmonary artery pressure
11. arterial-blood oxygen saturation
12. fraction of oxygen in inspired gas
13. ventilation pressure
14. carbon-dioxide content of expired gas
15. minute volume, measured
16. minute volume, calculated
17. hypovolemia
18. left-ventricular failure
19. anaphylaxis
20. insufficient anesthesia or analgesia
21. pulmonary embolus
22. intubation status
23. kinked ventilation tube
24. disconnected ventilation tube
25. left-ventricular end - diastolic volume
26. stroke volume
27. catecholamine level
28. error in heart rate reading due to low cardiac output
29. true heart rate
30. error in heart rate reading due to electrocautery device
31. shunt
32. pulmonary-artery oxygen saturation
33. arterial carbon-dioxide content
34. alveolar ventilation
35. pulmonary ventilation
36. ventilation measured at endotracheal tube
37. minute ventilation measured at the ventilator

Figure 4.1: The ALARM network.

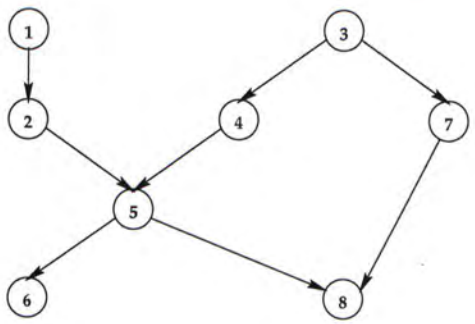


1. Spool Process OK
2. Local Disk Space Adequate
3. Application Output OK
4. Spooled Data OK
5. Print Spooling On
6. GDI Data Input OK
7. Correct Driver
8. Uncorrupted Driver
9. Correct Driver Settings
10. GDI Data Output OK
11. Correct Printer Selected
12. Correct Local Port
13. Local Cable Connected
14. Network Up
15. Correct Printer Path
16. Network Cable Connected
17. Print Data OK
18. Network Path OK
19. Network/Local Printing
20. Local Path OK
21. PC to Printer Transport OK
22. Printer On and Online
23. Paper Loaded
24. Printer Memory Adequate
25. Printer Data OK
26. Printer Output OK

Figure 4.2: The PRINTD network.

the authors), are implemented in the C++ language and are compiled using the same compiler [2]. Besides, the same MDL metric evaluation routine is used so that the difference among implementations are minimized. To enhance efficiency, all algorithms are implemented with the same hashing mechanism so that each computed MDL metric query is stored in a hash table. By a *query*, we refer to the evaluation of the MDL score of the given parent set of a node. For all of the algorithms, the allowable parent set size is limited to be five. Since the algorithms are stochastic in nature, they are executed 40 times for each testing instance. All our experiments are conducted on the Sun Ultra-5 workstations.

[2]We use the g++ compiler with "-O2" optimization level.



1. Visit to Asia
2. Tuberculosis
3. Smoking
4. Lung Cancer
5. Tuberculosis or Cancer
6. X-Ray Result
7. Bronchitis
8. Dyspnea

Figure 4.3: The ASIA network.

In the following sections, we shall present our experimental results.

4.2 Comparing the Learning Algorithms

In this section, we compare the performance of different algorithms on all of the data sets. We begin by comparing CCGA with MDLEP, then HEP with MDLEP, and end by contrasting our proposed algorithms. Our objective in these experiments is to demonstrate whether our proposed algorithms are more efficient than MDLEP. We estimate the performance of different algorithms using five measures, which include:

* the average MDL score of the final solutions, the smaller the better (AFS),
* the average MDL score of the best network obtained in the first generation (AIS),
* the average execution time in seconds (AET),
* the average generation that the best-so-far solution is obtained (ANG),
* the average number of MDL metric evaluations in a run (AME),
* the average structural difference, i.e. number of edges added, omitted and reversed, between the final solution and the original network (ASD).

Recalls that the algorithms are executed forty times for each data set, the figures are, therefore, an average of forty trials. Since most of the measures are self-explanatory, we will not discuss about the details. However, there are a two points that we like to clarify:

- AME is *not* the count of MDL metric evaluations invoked during a run. Since

all algorithms use the same hashing mechanism (i.e. store every computed query) for MDL metric evaluation, counting the number of invocations to the MDL evaluation function does not have much significance. A better way, therefore, is to record the number of stored queries which is a count of all distinct MDL metric evaluations that have taken place. This is what AME refers to.

- The structural difference statistics (i.e. ASD) is served primarily as a subsidiary measure. Because we use the MDL metric as the evaluation function, only by the score of the final solution obtained could we judge the fruitfulness of the search algorithms. Thus, with a comparatively smaller structural difference, it may not be a direct implication that an algorithm outperforms another. Nevertheless, it gives an estimate of our real concern: "Can the learning algorithm recover the original network structure?"

Without no finetunning, we adopt the parameter values as the default setting:

- For MDLEP, we adopt the same parameter settings that appear in the original publication [75]: the population size is 50, the tournament size is seven, and the maximum number of generation is 5,000.

- For CCGA, the cutoff value for the CI test phase is 0.3. For each species population in search phase, the population size is 20 with the crossover and mutation rate set to 0.7 and 0.2 respectively. The belief factor is 0.2. We use 1,000 generations as the termination criterion.

- For HEP, all parameter settings are identical to MDLEP with the additional parameter, Δ_α, set to be 0.02.

We note that it is difficult to provide a common ground to compare, in particular, CCGA with the EP formulations, because they are intrinsically different. One thing which would seem unfair is that CCGA uses a much larger search population (in total) than the EP formulations. Although this may be an inherent advantage of CCGA, we must not forget that CCGA only maintains a single search point (the collaborative structure), which could be a drawback. Given that the algorithms

use different search methodologies, our principal for selecting the values is based on what we would regard as a "normal setting."

4.2.1 Comparing CCGA with MDLEP

We provide a summary of the results in Table 4.2. In the table, the MDL score of the original network is shown under the name of the data set for reference. Besides the averaged measure, we also include the standard deviations of the respective measure which appear in parentheses.

Data Set		AFS	AIS	AET	ANG	AME	ASD
ALARM-1000 (18533.5)	CCGA	17,877.3 (38.5)	23,527.7 (885.6)	44.9 (1.1)	398.6 (290.7)	5,978.2 (110.1)	12.6 (2.3)
	MDLEP	17,990.5 (73.1)	30,831.0 (795.6)	1,003.9 (70.8)	4,301.2 (654.3)	22,133.8 (619.3)	19.4 (4.2)
ALARM-2000 (34287.9)	CCGA	33,836.5 (92.8)	45,720.0 (1,750.3)	68.1 (1.7)	384.4 (265.6)	8,710.8 (139.9)	8.2 (1.2)
	MDLEP	33,932.6 (215.8)	56,896.6 (1,259.5)	1,307.8 (125.1)	4,046.6 (634.1)	25,905.8 (911.3)	12.9 (4.9)
ALARM-5000 (81233.4)	CCGA	81,033.1 (64.4)	111,738.0 (4,389.9)	114.1 (1.5)	422.1 (264.9)	9,118.1 (139.5)	6.1 (0.5)
	MDLEP	81,287.6 (419.9)	134,487.2 (1,836.0)	1,843.2 (359.0)	3,946.3 (651.2)	29,570.8 (1,016.3)	10.7 (4.9)
ALARM-10000 (158497.0)	CCGA	158,432.4 (16.3)	224,246.3 (7,070.0)	204.6 (3.6)	422.8 (286.8)	10,531.5 (96.2)	3.4 (0.7)
	MDLEP	158,704.4 (513.1)	256,946.2 (3,843.7)	2,435.1 (350.1)	3,596.7 (720.0)	32,160.8 (1,538.0)	8.7 (5.1)
ALARM-O (138455.0)	CCGA	138,854.3 (564.1)	217,386.1 (12,898.4)	269.8 (32.5)	462.7 (247.1)	13,635.7 (186.4)	11.9 (5.0)
	MDLEP	138,913.4 (460.8)	252,818.4 (5,862.0)	4,209.9 (2,021.3)	4,523.8 (482.1)	34,309.5 (1,327.5)	17.5 (6.9)
ASIA-1000 (3416.9)	CCGA	3,413.4 (0.0)	3,650.7 (104.1)	2.9 (0.1)	5.9 (5.3)	42.1 (0.5)	3.7 (0.5)
	MDLEP	3,398.6 (0.0)	3,590.2 (48.5)	76.3 (0.4)	79.6 (30.2)	656.8 (9.2)	3.5 (0.5)
PRINTD-5000 (106541.6)	CCGA	106,541.6 (0.0)	114,967.2 (900.3)	66.4 (7.5)	10.1 (3.9)	2,552.1 (43.8)	0.0 (0.0)
	MDLEP	106,541.6 (0.0)	116,089.6 (546.4)	704.5 (13.8)	512.1 (95.8)	17,688.4 (373.7)	0.0 (0.0)

Table 4.2: Performance comparison between CCGA and MDLEP

Except for the ASIA-1000 data set, wee can observe that CCGA is often able to find better or equally good network comparing with MDLEP. For two out of the six cases, the difference is statistically significant at the 0.05 level using the Mann-Whitney test[3]. Using the MDL score of the original network as a reference, we

[3]Mann-Whitney test is a non-parametric test which suits our need as the final score is observed

observe that although MDLEP performs well (competitive with CCGA) for smaller data sets, it clearly needs longer running time to compete with our approach for larger data sets. For the ALARM-O data set, we regard it as a harder problem instance. Even the size of data set is relatively large, both algorithms fail to approximate the score of the original network. However, CCGA still have a better performance. For the PRINTD-5000 data set, both algorithms could recover the original network structure and hence the two have identical performance. For the ASIA-1000 data set, we find that MDLEP outperforms CCGA in terms of the final score. On closer inspection, we find that the reason is because an important edge in the network has been cut away during the CI test phase. Consequently, CCGA is stuck at a local optimal solution. In another setting, when we set the cutoff value to one (i.e. ignore the test results), we find that the performance of CCGA will equal MDLEP.

Regarding the structural difference measure (i.e. ASD), we observe that CCGA consistently performs better than MDLEP. There are two possibilities which account for the observation: one directly relates with the CI tests, another relates indirectly. On the one hand, the CI test phase possibly helps to focus the search on dealing with only the *correct* edges (that appears in the original structure) so that the result returned will be similar to the original one. Although MDLEP may be successful in finding structures with low scores, such structures may contain some *wrong* edges. Hence, it will be the merit of the entire hybrid learning framework which helps to recover structures that closely resemble the original one. On the other hand, the observation could also be explained by that better results are obtained because searching is efficient. In this regard, we assume that the metric directly relates with the structural difference measure. Hence, networks with good scores also will resemble the original network. Because the searching is made efficient as a consequence of the reduction of search space by CI tests, we can often find these good solutions so that ASD is small.

Apart from the quality of the final solutions, we observe that CCGA has a tremendous speedup over MDLEP. In general, the gain is more than ten-fold (varies from 10.6 to 26.5). We conjecture that there are two important reasons: (1) Due

not to follow a normal distribution.

to the hybrid framework, the search space is reduced. Therefore, CCGA issues significantly less MDL metric evaluations (i.e. AME) than MDLEP, which is crucial as to evaluate the MDL metric is a time-consuming operation [4]. Despite that less evaluation is made, CCGA is still effective in finding good solutions. (2) Besides, because CCGA requires much less cycle repairing operations (only for the collaborative structure and the merged result) than MDLEP, it could also have saved much time.

Because that CCGA executes faster while the result can still rival MDLEP, we conclude that CCGA is more efficient than MDLEP.

4.2.2 Comparing HEP with MDLEP

We perform a similar analysis to compare HEP with MDLEP. We summarize our results in Table 4.3.

Data Set		AFS	AIS	AET	ANG	AME	ASD
ALARM-1000 (18533.5)	HEP	17880.56 (31.9)	24323.5 (1,186.6)	204.75 (3.9)	817.6 (1,163.0)	3282.5 (1,061.8)	11.15 (2.3)
	MDLEP	17,990.5 (73.1)	30,831.0 (795.6)	1,003.9 (70.8)	4,301.2 (654.3)	22,133.8 (619.3)	19.4 (4.2)
ALARM-2000 (34287.9)	HEP	33777.8 (62.9)	44199.45 (1,324.9)	225.63 (10.0)	1410.78 (1,540.2)	5332.05 (2,601.7)	9.05 (1.4)
	MDLEP	33,932.6 (215.8)	56,896.6 (1,259.5)	1,307.8 (125.1)	4,046.6 (634.1)	25,905.8 (911.3)	12.9 (4.9)
ALARM-5000 (81233.4)	HEP	81004 (0.0)	102310.02 (2,352.0)	290.3 (11.9)	448.57 (796.0)	4826.57 (2,229.5)	6.05 (0.5)
	MDLEP	81,287.6 (419.9)	134,487.2 (1,836.0)	1,843.2 (359.0)	3,946.3 (651.2)	29,570.8 (1,016.3)	10.7 (4.9)
ALARM-10000 (158497.0)	HEP	158498.5 (298.5)	199210.75 (5,082.8)	384.77 (27.5)	970.42 (879.4)	6183.5 (3,061.1)	4.53 (2.8)
	MDLEP	158,704.4 (513.1)	256,946.2 (3,843.7)	2,435.1 (350.1)	3,596.7 (720.0)	32,160.8 (1,538.0)	8.7 (5.1)
ALARM-O (138455.0)	HEP	138,693.4 (538.1)	182,363.7 (5,856.5)	344.2 (23.7)	1,179.3 (1,250.1)	5,292.8 (3,032.8)	9.5 (5.7)
	MDLEP	138,913.4 (460.8)	252,818.4 (5,862.0)	4,209.9 (2,021.3)	4,523.8 (482.1)	34,309.5 (1,327.5)	17.5 (6.9)
ASIA-1000 (3416.9)	HEP	3398.99 (0.7)	3455.74 (7.3)	39.34 (1.0)	18.93 (7.1)	166.85 (18.6)	2.25 (1.0)
	MDLEP	3,398.6 (0.0)	3,590.2 (48.5)	76.3 (0.4)	79.6 (30.2)	656.8 (9.2)	3.5 (0.5)
PRINTD-5000 (106541.6)	HEP	106,541.6 (0.0)	111382.88 (407.3)	172.85 (3.6)	38.15 (10.4)	1813.33 (453.6)	0 (0.0)
	MDLEP	106,541.6 (0.0)	116,089.6 (546.4)	704.5 (13.8)	512.1 (95.8)	17,688.4 (373.7)	0.0 (0.0)

Table 4.3: Performance comparison between HEP and MDLEP

[4] In our current implementation, every record needs to be examined once for each query.

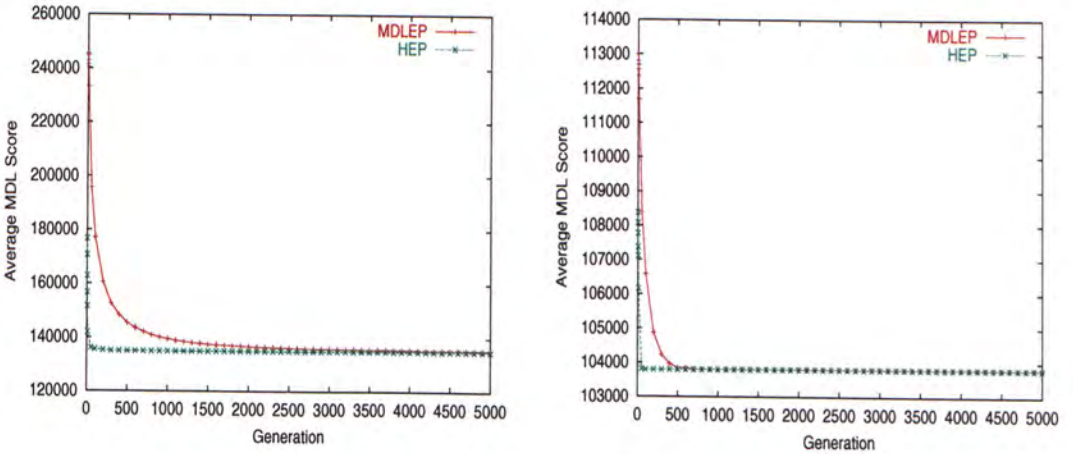
It is observed that HEP can in general find better or equally good solutions than MDLEP for all the data sets. Apart from the ASIA-1000 and the PRINTD-5000 data set, the difference is statistically significant at the 0.05 level for the rest of data sets. Note that the problem that persists in CCGA for the ASIA-1000 data set does not appear for HEP. This is as expected because HEP adopts a varying cutoff value instead of a fixed one. If we compare the ANG statistics, it is found that HEP uses much less generations to obtain the final solution (statistically significant at the 0.05 level using a one-tailed t-test). Given that HEP and MDLEP essentially use the same formulation in searching, the experimental results readily suggest that HEP is more efficient as it uses fewer generations to obtain similar, or better, solutions.

Similar to CCGA, it is observed that HEP performs better than MDLEP in terms of ASD.

Under the same termination criterion, HEP uses less time to finish than MDLEP (i.e. from the AET statistics). Despite that the time spent on the CI phase is also counted in HEP, we can easily deduce that a *generation* in HEP takes less time than in MDLEP. But if similar operations (i.e. mutations) are performed for the same amount of individuals, what constitutes a speedup? An explanation for this is due to the prevention of cycle formation in HEP. From empirical experience, cycle repairing in HEP often takes a significant portion of the total running time. By avoiding to create illegal network (i.e. that contains cycle), HEP does not need to spend time on cycle repairing for every offspring as MDLEP does. Although there is overhead involved (see Section 3.4.3), it seems that the result is in favor of the saying, "prevention is better than cure." Another possible reason for HEP being faster is because HEP issues fewer MDL metric evaluations. When comparing the count of the metric evaluations (AME), we observe that HEP makes orders of magnitude less evaluations than MDLEP. The could be accounted by the reduction of the search space due to the hybrid framework.

If we compare the AIS statistics, it is clear that HEP could often have a better starting point than MDLEP. Apparently, this is also the benefit that the hybrid framework brings as we take CI test result into consideration rather than to initialize the population randomly.

In Figure 4.4, we compare the typical runs of both algorithms for the two data



(a) ALARM-O data set

(b) PRINTD-5000 data set

Figure 4.4: Typical runs of both algorithms.

set, ALARM-O and PRINTD-5000. For each algorithm, we measure the MDL score of the best-so-far solution average over forty runs as generations proceed. Although we are testing on two different data sets, we obtain similar observation that HEP converges much faster than MDLEP to the final solution. Besides, for the same number of generations, HEP is observed to perform better than MDLEP in terms of the average score of the final solution obtained.

To validate our claim that HEP improves over MDLEP by having more better offspring produced, we obtain the result as shown in Figure 4.5. The testing data set used is the ALARM-O data set [5]. For both algorithms, we record the number of better offspring (i.e. better than its parent) produced at each generation averaged over forty runs. From the figure, we can clearly observe that HEP is able to produce more better offspring than MDLEP. In addition, after a number of generations, we observe that MDLEP could rarely produce better offspring. On the other hand, because of the merge operator, HEP still manages to produce a fair amount of better offspring. We give a detailed analysis on the effectiveness of the merge operator in Section 4.4.4.

[5] Although we do not the result, a similar observation is obtained for the PRINTD data set.

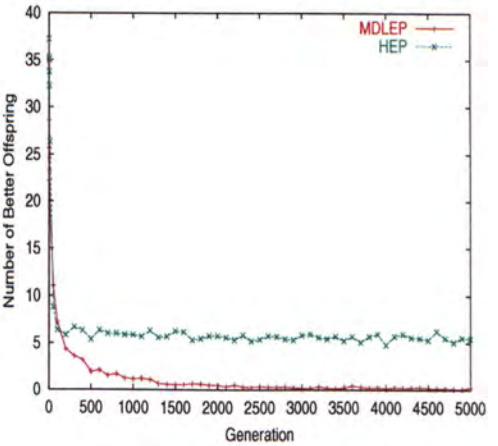


Figure 4.5: Comparing the number of better offspring produced.

4.2.3 Comparing CCGA with HEP

After demonstrating that CCGA and HEP are both more efficient than MDLEP, it is of our interest to compare CCGA with HEP. In general, we claim that neither algorithms are superior to one another. Recalling that both algorithms use different search methodology, it is not surprising that each of them would excel for particular situations. Here, we make the comparison between the two algorithms by reproducing the previous findings. We summarize the result in Table 4.4.

From the table, we can observe that both algorithms perform similarly in terms of the average score of the final solution (AFS). Even when difference exists, the magnitude is often small. For some case (i.e. ALARM-2000, ALARM-5000, ASIA-1000), HEP performs better (statistically significant at the 0.05 level). On the other hand, for the ALARM-10000 data set, CCGA performs better (statistically significant at the 0.05 level). For the rest, the difference between the two algorithms are not statistically significant.

Recalling that CCGA uses a fixed α while HEP does not, a difference in performance could possibly be explained by the trade-off involved. Hence, if the choice of α in CCGA (0.3 in all the cases) is ideal for a particular data set such that only crucial edges are left, CCGA will be more efficient than HEP. On the other hand, if the choice of α is improper, CCGA may get stuck at a local optima, which is what we observe in the case of ASIA-1000.

Data Set		AFS	AIS	AET	ANG	AME	ASD
ALARM-1000 (18533.5)	CCGA	17,877.3 (38.5)	23,527.7 (885.6)	44.9 (1.1)	398.6 (290.7)	5,978.2 (110.1)	12.6 (2.3)
	HEP	17880.56 (31.9)	24323.5 (1,186.6)	204.75 (3.9)	817.6 (1,163.0)	3282.5 (1,061.8)	11.15 (2.3)
	MDLEP	17,990.5 (73.1)	30,831.0 (795.6)	1,003.9 (70.8)	4,301.2 (654.3)	22,133.8 (619.3)	19.4 (4.2)
ALARM-2000 (34287.9)	CCGA	33,836.5 (92.8)	45,720.0 (1,750.3)	68.1 (1.7)	384.4 (265.6)	8,710.8 (139.9)	8.2 (1.2)
	HEP	33777.8 (62.9)	44199.45 (1,324.9)	225.63 (10.0)	1410.78 (1,540.2)	5332.05 (2,601.7)	9.05 (1.4)
	MDLEP	33,932.6 (215.8)	56,896.6 (1,259.5)	1,307.8 (125.1)	4,046.6 (634.1)	25,905.8 (911.3)	12.9 (4.9)
ALARM-5000 (81233.4)	CCGA	81,033.1 (64.4)	111,738.0 (4,389.9)	114.1 (1.5)	422.1 (264.9)	9,118.1 (139.5)	6.1 (0.5)
	HEP	81004 (0.0)	102310.02 (2,352.0)	290.3 (11.9)	448.57 (796.0)	4826.57 (2,229.5)	6.05 (0.5)
	MDLEP	81,287.6 (419.9)	134,487.2 (1,836.0)	1,843.2 (359.0)	3,946.3 (651.2)	29,570.8 (1,016.3)	10.7 (4.9)
ALARM-10000 (158497.0)	CCGA	158,432.4 (16.3)	224,246.3 (7,070.0)	204.6 (3.6)	422.8 (286.8)	10,531.5 (96.2)	3.4 (0.7)
	HEP	158,498.5 (298.5)	199,210.75 (5,082.8)	384.77 (27.5)	970.42 (879.4)	6,183.5 (3,061.1)	4.53 (2.8)
	MDLEP	158,704.4 (513.1)	256,946.2 (3,843.7)	2,435.1 (350.1)	3,596.7 (720.0)	32,160.8 (1,538.0)	8.7 (5.1)
ALARM-O (138455.0)	CCGA	138,854.3 (564.1)	217,386.1 (12,898.4)	269.8 (32.5)	462.7 (247.1)	13,635.7 (186.4)	11.9 (5.0)
	HEP	138,693.4 (538.1)	182,363.7 (5,856.5)	344.2 (23.7)	1,179.3 (1,250.1)	5,292.8 (3,032.8)	9.5 (5.7)
	MDLEP	138,913.4 (460.8)	252,818.4 (5,862.0)	4,209.9 (2,021.3)	4,523.8 (482.1)	34,309.5 (1,327.5)	17.5 (6.9)
ASIA-1000 (3416.9)	CCGA	3,413.4 (0.0)	3,650.7 (104.1)	2.9 (0.1)	5.9 (5.3)	42.1 (0.5)	3.7 (0.5)
	HEP	3398.99 (0.7)	3455.74 (7.3)	39.34 (1.0)	18.93 (7.1)	166.85 (18.6)	2.25 (1.0)
	MDLEP	3,398.6 (0.0)	3,590.2 (48.5)	76.3 (0.4)	79.6 (30.2)	656.8 (9.2)	3.5 (0.5)
PRINTD-5000 (106541.6)	CCGA	106,541.6 (0.0)	114,967.2 (900.3)	66.4 (7.5)	10.1 (3.9)	2,552.1 (43.8)	0.0 (0.0)
	HEP	106,541.6 (0.0)	111382.88 (407.3)	172.85 (3.6)	38.15 (10.4)	1813.33 (453.6)	0 (0.0)
	MDLEP	106,541.6 (0.0)	116,089.6 (546.4)	704.5 (13.8)	512.1 (95.8)	17,688.4 (373.7)	0.0 (0.0)

Table 4.4: Performance comparison between CCGA and HEP.

The AME statistics, which estimates how many different solutions are examined during a run, shows that CCGA explores a larger search space than HEP. This is as expected because CCGA uses a much larger search population in total. But still, both CCGA and HEP issue far less MDL metric evaluations than MDLEP.

Although CCGA explores a larger search space, we observe that it often uses less time than HEP (AET). A possible reason is due to the efficiency of manipulating bit-strings and the single-point search strategy. In CCGA, besides the collaborative

structure and the best-so-far structure which exist as a complete network, most of the manipulations operate on bit-strings. Because the operations are simple, CCGA thus could be faster than HEP provided that such gain is not outweighed by the overhead of exploring a larger search space. We provide more insight into the problem in Section 4.3.1.

4.3 Performance Analysis of CCGA

In this section, we study the performance of CCGA under different settings. This include varying α value, varying population size, varying crossover and mutation probabilities, and varying belief factor. For all the experiments, we use the ALARM-O data set for testing as we regard it as a difficult problem instance and a difference in performance could readily be observed. To focus on the difference due to settings, we use the same set of random seed for the forty trials which guarantee the initial populations are the same, if possible. To compare the performance, we use the same measures mentioned above. Furthermore, we assume the following default setting: the cutoff value for the CI test phase is 0.3; the termination criterion is 1,000 generations; the population size for each species is 20; the belief factor is 0.2; the crossover and mutation rate set to 0.7 and 0.2 respectively.

4.3.1 Effect of Different α

Although we argue in Section 3.4.1 that the choice of α can have a critical impact there is some common consensus in interpreting the p-value, and hence the choice of α. For instance, according to [5], different p-values will have the following meanings (with the null hypothesis referring to two nodes being conditionally independent):

p-value	Interpretation
≈ 0.01	The null hypothesis is almost certainly false.
≈ 0.025	Serious doubt about the truth of the null hypothesis
≈ 0.05	Some doubt about the truth of null hypothesis
$\gg 0.1$	No evidence to doubt the truth of the null hypothesis

Table 4.5: Interpretation of the p-value

Since α determines whether an hypothesis is to be rejected, its choice will lead to the following interpretations:

α value	Interpretation
≈ 0.01	Reject the null hypothesis when it is almost certainly false.
≈ 0.025	Reject the null hypothesis even if there is only serious doubt about its truth
≈ 0.05	Reject the null hypothesis even if there is only some doubt about its truth
$\gg 0.1$	Reject the null hypothesis even when there seems no evidence to doubt its truth

Table 4.6: Interpretation of α

Although this table provides a guideline for choosing α, we note that the advice may not be useful. Our primary concern is to make as few error as possible such that we do not admit a conditional independence relation wrongly. This would suggest that we use a larger value of α. Meanwhile, our secondary concern is to know the limit (i.e. smallest possible) that we would benefit most from the CI test phase (i.e. prune away every irrelevant edges). While the table only presents a general and vague meaning of a choice, it does not provide any hint to what we want. On the other hand, because different data sets have different probability distributions, the credibility of the testing result is likely to be problem dependent. Thus, we argue that only through empirical testing could we know which α is optimal.

In this experiment, our objective is to demonstrate the effect of using different values of α. In particular, we test the values of 0.02, 0.05, 0.3, 0.5 and 1.0. Since a larger α value implies that the null hypothesis is more easier to be rejected, it is expected that the reduction in search space will be less for increasing α value. In other words, the benefit from the CI test phase is diminishing. For the extreme case that α equals 1.0, it is equivalent to omitting the CI test phase from the hybrid framework. We summarize our findings in Table 4.7.

These results coincide with our expectation that smaller value of α will be more effective. By using a smaller value, the running time is significantly shorter. As an example, the time used for $\alpha = 0.02$ is only one-third of the time used for $\alpha = 0.5$. By using a smaller value, we focus on a smaller value of interest in the entire search space. Thus the search could be more efficient. The AME statistics, which measures how many different solutions are examined, is supportive of our argument.

Although we expect that a smaller value of α will be prone to erroneous CI test

	AFS	AIS	AET	ANG	AME	ASD
$\alpha = 0.02$	138,712.4 (356.7)	203,667.3 (11,446.4)	173.0 (23.0)	448.9 (216.1)	4,096.4 (62.5)	11.0 (4.3)
$\alpha = 0.05$	138,866.5 (528.2)	201,723.3 (11,651.8)	191.6 (23.9)	512.4 (272.9)	5,189.3 (85.3)	12.4 (5.1)
$\alpha = 0.3$	138,854.3 (564.1)	217,386.1 (12,898.4)	269.8 (32.5)	462.7 (247.1)	13,635.7 (186.4)	11.9 (5.0)
$\alpha = 0.5$	138,858.4 (642.8)	223,194.1 (13,842.5)	497.7 (37.5)	523.8 (293.2)	37,160.7 (878.5)	12.7 (4.3)
$\alpha = 1.0$	141,456.6 (893.4)	215,187.5 (11,235.0)	6,664.5 (68.7)	904.7 (108.8)	814,931.6 (3,397.9)	39.1 (4.8)

Table 4.7: Performance of CCGA with different α.

result (which is what we observed in the ASIA-1000 data set), we do not observe the problem to be consequential in the experiment. This suggests that the issue is likely to be dependent on the data set. Despite that the time used increases with increasing α, we notice that the final solution we obtain is similar for the choice of α ranging between 0.05 to 0.5 (Kruskal-Wallis test[6] returns p-value $= 0.43$). Hence, CCGA seems to be robust when our choice of α is moderate.

For the extreme case that $\alpha = 1.0$, the performance of CCGA is poor. Without the help from CI test, CCGA fails to find good solutions with the same termination criterion. Beside, CCGA is also extremely slow, which is a result of extensive exploration of the search space. Thus, it is evident that the hybrid framework plays an important role in CCGA.

4.3.2 Effect of Different Population Sizes

Often, population size plays an important role in evolutionary computation. In general, by using a larger population size, it is more likely that a good solution could be encountered early. However, the price is that more computations will be needed for each generation. Thus, if we can strike the balance, the performance of our algorithm can be optimized.

In this experiment, we compare the performance of CCGA for different population sizes. With other parameters fixed, we increase the population size, m, in step of ten, from 20 up to 50. We put on findings in Table 4.8.

The overhead of using a larger population is manifest in the findings: the average

[6]Kruskal-Wallis test is an extension of Mann-Whitney test for multiple groups analysis.

	AFS	AIS	AET	ANG	AME	ASD
$m = 20$	138,854.3 (564.1)	217,386.1 (12,898.4)	269.8 (32.5)	462.7 (247.1)	13,635.7 (186.4)	11.9 (5.0)
$m = 30$	138,631.1 (162.3)	214,091.9 (10,703.8)	294.3 (31.6)	466.5 (260.1)	16,034.3 (325.6)	10.3 (4.3)
$m = 40$	138,710.8 (452.6)	213,138.4 (12,088.9)	310.7 (28.4)	396.3 (218.8)	17,887.4 (310.2)	11.6 (4.0)
$m = 50$	138,676.0 (427.4)	212,518.9 (10,641.6)	331.1 (28.3)	406.5 (268.5)	19,682.3 (373.9)	10.5 (4.2)

Table 4.8: Performance of CCGA under different population sizes.

execution time (AET) increases with increasing population size. As a consequence of having more search points, the average number of MDL metric evaluations (AME) also increases. Since the initial structure is generated by assembling representatives from species population, having a larger population results in a higher chance of getting a better representative. Hence, the initial structure is observed to have a better score for larger population (AIS).

Although we expect that by increasing the population size, we could also accelerate the searching (in terms of number of generations), the result does not show much evidence supporting this claim. If we compare ANG for different population sizes, the largest difference is of a small magnitude (i.e. about 70 generations). The Kruskal-Wallis test also suggests that the differences is not statistically significant (p-value equals 0.451). In other words, larger population size does not help to obtain the final solution earlier. Besides, the final results (AFS) are also similar. Again, the Kruskal-Wallis test suggests that the difference is not statistically significant (p-value equals 0.490). Thus, while the advantage of using a larger population is not apparent, we recommend the choice of a smaller population size.

4.3.3 Effect of Varying Crossover and Mutation Probabilities

Apart from the population size, we also investigate the effect of various crossover and mutation probabilities combinations. In particular, we test the performance of CCGA with the crossover probability, denoted by p_c, taking the values of 0.5, 0.7 and 0.9 while the mutation probability, p_m, is varied among 0.05, 0.2 and 0.5. Hence, we have tried a total of nine combinations. We present our findings in Table 4.9.

Since we are using the same set of random seeds, the average initial scores (AIS) are the same and are therefore omitted from the table.

	AFS	AET	ANG	AME	ASD
$p_c = 0.5$	138,716.8 (453.1)	221.8 (23.6)	419.5 (250.0)	8,288.0 (163.6)	10.6 (4.3)
$p_c = 0.7$	138,925.9 (737.7)	218.4 (27.5)	471.0 (238.8)	8,106.2 (152.7)	11.7 (4.5)
$p_c = 0.9$	138,815.9 (567.8)	210.4 (24.5)	559.4 (253.3)	7,857.7 (140.9)	11.5 (4.5)

(a) $p_m = 0.05$

	AFS	AET	ANG	AME	ASD
$p_c = 0.5$	138,952.9 (662.1)	270.5 (19.0)	547.0 (277.3)	14,194.0 (174.4)	12.9 (4.4)
$p_c = 0.7$	138,854.3 (564.1)	269.8 (32.5)	462.7 (247.1)	13,635.7 (186.4)	11.9 (5.0)
$p_c = 0.9$	139,059.7 (796.3)	253.3 (20.0)	485.3 (247.2)	13,144.2 (213.5)	13.3 (3.2)

(b) $p_m = 0.2$

	AFS	AET	ANG	AME	ASD
$p_c = 0.5$	139,136.3 (647.7)	353.9 (31.7)	563.1 (231.3)	21,737.3 (455.9)	14.5 (4.0)
$p_c = 0.7$	138,978.3 (560.1)	335.1 (28.4)	496.3 (264.4)	21,292.4 (396.1)	13.1 (4.5)
$p_c = 0.9$	138,953.1 (527.8)	333.8 (30.3)	557.2 (244.3)	20,729.6 (371.8)	13.1 (4.1)

(c) $p_m = 0.5$

Table 4.9: Performance of CCGA with different p_c and p_m.

From the table, we can observe that the count of MDL metric evaluation (AME) increases with increasing mutation probability. On the other hand, the difference caused by changes in crossover probability is order of magnitude less than that caused by changes in mutation probability. As AME reflects the number of distinct solutions examined, this coincides with our general expectation that more mutations will lead to the creation of more new solutions. As a consequence of more evaluations, the execution time also increases. This is supported by the result we obtained. We illustrate these observations in Figure 4.6.

Although the choice of the crossover probability seems to exhibit little impact on the performance, it is observed that higher crossover probability consistently decreases the number of MDL metric evaluation and hence the running time. To account for this, we notice that a higher crossover probability also implies a larger

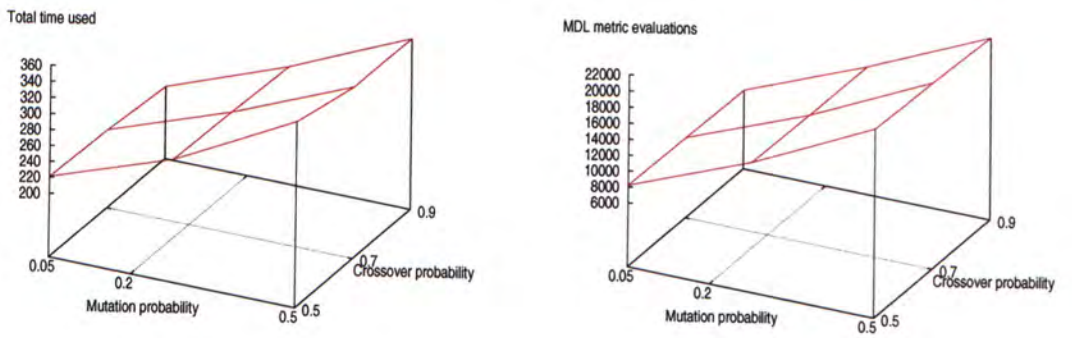


Figure 4.6: Effects on time used and number of evaluations.

portion of the population (inside a species population) undergo gene exchange. In such case, a super-fit individual will likely to dominate the population quickly. Thus, more individuals are alike and the population converges in a few generations. As an example, suppose that a super-fit individual, which encodes a parent set that is sub-optimal, appears in the population. The sub-optimal parent set will spread around in the population quickly as a result of more crossover. Thus, most individuals will be similar and essentially encode the similar subset of parents. As a result, there will be less distinct individuals produced in the long run in compared with the case for lower crossover probability where diversity is more likely to be preserved for a longer time.

If we look at the result (AFS and ANG) obtained in different runs, we notice that there are no significant differences among them. Furthermore, there is no observable trend that we could follow. For the seemingly superior combination of $p_c = 0.5$ and $p_m = 0.05$, the superiority seems to be vulnerable: for the three runs using $p_m = 0.05$, the Kruskal-Wallis test suggests that the differences in the final score obtained are not statistically signficiant (p-value equals 0.394). In conclusion, the experimental result favors the choice of using a low mutation probability while the crossover probability should not be too high.

4.3.4 Effect of Varying Belief Factor

In essence, the belief factor is the minimum threshold that we believe an edge, which appears in the collaborative structure $\mathcal{S}$, is correct (See Section 3.3.2). In this experiment, we test CCGA using different values of belief factor, which include 0.0, 0.1, 0.2, 0.5, 0.7 and 1.0. We present the result in Table 4.10.

	AFS	AET	ANG	AME	ASD
factor = 0.0	138,761.9 (461.7)	559.5 (106.8)	498.1 (250.0)	12,519.5 (255.2)	11.6 (4.0)
factor = 0.1	138,761.1 (467.7)	352.8 (52.0)	497.2 (291.4)	13,688.9 (212.9)	11.9 (3.9)
factor = 0.2	138,854.3 (564.1)	269.8 (32.5)	462.7 (247.1)	13,635.7 (186.4)	11.9 (5.0)
factor = 0.5	139,286.0 (731.9)	231.0 (8.5)	590.0 (265.2)	12,004.1 (255.2)	15.5 (3.7)
factor = 0.7	140,052.6 (826.2)	233.3 (11.9)	626.3 (281.1)	10,160.9 (293.8)	19.4 (3.8)
factor = 1.0	164,003.6 (5,972.5)	107.7 (14.9)	12.7 (9.8)	1,799.4 (705.2)	41.5 (4.3)

Table 4.10: Performance comparison of different belief factor.

In the extreme case that the belief factor equals 1.0, we believe in the correctness of every existing and updated edges. Thus, the topological ordering given by $\mathcal{S}$, and hence the constraints, are unchanged. CCGA will therefore evolve an optimal structure for an initial ordering which is likely to be a sub-optimal solution. This explains the poor performance of CCGA when the belief factor equals 1.0.

As the belief factor increases, the average MDL metric evaluation (AME) drops. This is reasonable as a smaller belief factor means we have more doubts about the placement of an edge. If more edges seem to be wrong, we have fewer constraints. Therefore, the search space is large when compared to the one using a larger belief factor. Consequently, this also explains the drop in the execution time when the belief factor increases.

Meanwhile, a larger belief factor seems to deliver poorer final solution (evident when the belief factor varies from 0.2 to 0.5). Hence, despite that a larger belief factor would quicken up CCGA, it has the adverse effect of leading us to a sub-optimal solution. This states the trade off involved in choosing a large belief factor. Nevertheless, observing that there is no statistically significant degradation of the final solution obtained when the belief factor increase from 0.0 to 0.2 (p-value equals

0.246), the adopted value of 0.2 seems to be a fairly good choice.

4.4 Performance Analysis of HEP

In the same vein, we study the factors that affect the performance of HEP. Particularly, there are four issues that we wish to investigate: (1) the contribution due to the hybrid framework and the merge operator, (2) the effect of the population size, (3) the effect of $\Delta\alpha$, and (4) the efficiency of the merge operator. Similar to the previous study on CCGA, we conduct all our experiments on the ALARM-O data set and use the same set of random seed for the forty trials. We assume the following default setting: the termination criterion is 5,000 generations; the population size is 50; the tournament size is 7; Δ_α is 0.02.

4.4.1 The Hybrid Framework and the Merge Operator

Recall that we have proposed three strategies to improve over MDLEP, it is important to investigate the actual merits of each of them. As the experimental result suggests, the cycle prevention scheme seems to be indispensable because it contributes largely to the observed speedup. However, we note that the cycle prevention scheme is only an enhancement in implementation, which implies that it does not actually improve the efficiency as a better search methodology. Hence, our focus is on the comparison between the hybrid framework and the merge operator.

Before we present the comparison made, we first describe the experiment which demonstrates the effectiveness of the cycle prevention scheme. In table 4.11, we name the approach that has only cycle prevention (without the hybrid framework and the merge operator) as "CP+EP." When we compare the time used (AET) between MDLEP and CP+EP, it is evident that the cycle prevention scheme helps to provide a speedup. However, when we compare the quality of the final solutions (AFS), the difference is not statistically significant (p-value equals 0.184) which suggests that only similar quality solutions are obtained. Furthermore, as the ANG statistics suggest, CP+EP would require even longer generations to obtain the final solution. Thus, we conclude that the cycle prevention scheme merely helps to improve the

speed, yet provides no obvious improvement on the search methodology. On the other hand, if we compare CP+EP with HEP, we find that there are statistically significant differences between the quality of the final solutions (p-value equals 1.38×10^{-14}), the ANG (p-value equals 5.27×10^{-14}) and even the time used (p-value equals 0.002). Hence, we conclude that it is the hybrid framework and the merge operator that provides the actual improvement on the efficiency.

	AFS	AIS	AET	ANG	AME	ASD
MDLEP	138,913.4 (460.8)	252,818.4 (5,862.0)	4,209.9 (2,021.3)	4,523.8 (482.1)	34,309.5 (1,327.5)	17.5 (6.9)
CP+EP	139,109.6 (591.8)	253,895.1 (7,065.6)	517.7 (21.1)	4,853.8 (132.0)	57,000.5 (2,904.1)	20.5 (7.6)
HEP	138,693.4 (538.1)	182,363.7 (5,856.5)	344.2 (23.7)	1,179.3 (1,250.1)	5,292.8 (3,032.8)	9.5 (5.7)

Table 4.11: Effectiveness of the cycle prevention scheme.

Frankly, if either the hybrid framework or the merge operator is good enough, there is no need to bring them together. Hence, the objective of our experiment is to show the justification of the HEP framework. To conduct the experiment, we compare HEP with two other implementations that we call "EP+CI" and "EP+merge." In EP+CI, we keep the hybrid framework but we do not use the merge operator; while in EP+merge, we do not perform any CI tests but keep the use of the merge operator. We present our result in Table 4.12. Note that for both implementations, the cycle prevention scheme is in use.

	AFS	AIS	AET	ANG	AME	ASD
EP+merge	138,602.3 (309.7)	256,653.1 (4,874.4)	323.8 (5.7)	1,715.3 (1,407.7)	18,810.3 (752.9)	11.2 (5.3)
EP+CI	138,609.4 (606.5)	182,363.7 (5,856.5)	279.6 (12.2)	1,983.0 (1,295.9)	4,255.2 (1,167.5)	9.0 (5.2)
HEP	138,693.4 (538.1)	182,363.7 (5,856.5)	344.2 (23.7)	1,179.3 (1,250.1)	5,292.8 (3,032.8)	9.5 (5.7)

Table 4.12: Performance comparison of different implementations.

In comparing the average execution time of the three implementations, we observe there are clear distinctions among them. Since HEP combines both strategies, it naturally takes the longest running time. On the other hand, the hybrid framework alone implementation, EP+CI, takes the shortest time while EP+merge comes in between.

To compare the final solutions obtained, we present a more detailed analysis in Table 4.13. In the table, we show the best score obtain among the forty trials, with the number in the parenthesis counting the number of occurrence. Besides, the lower quartile is included for which better or equally good scores are obtained for 75% of the forty trials ($40 \times 75\% = 30$). The worst score is the worst one out of the forty trials while the average is the AFS measure shown previously in Table 4.12.

	Best Score	Lower quartile	Worst score	Average
EP+merge	138,275 (9)	138,687	139,409	138,602.3
EP+CI	138,275 (17)	138,668	141,865	138,609.4
HEP	138,275 (17)	138,692	140,265	138,693.4

Table 4.13: A comparison of the final scores obtained for different implementations.

From the table, different implementations seems to deliver similar performance in terms of the distribution roughly described. However, it is observed that EP+merge has a smaller chance of hitting the recorded-best solution although it has the lowest average score. On the other hand, although the other two implementations can obtain the recorded-best solution most of the time, they also have a higher chance to obtain worse solution (comparing the lower quartile and the worst score). From such observation, we conjecture that the presence of the hybrid framework could possibly result in the negligence of many near-optimal solutions as they contain the *wrong* edges. This would be an drawback to the confinement to a reduced search space although it accelerates the searching for an optimal solution.

Although the result is not suggestive of a particular recommendation, it must be said that both EP+merge and EP+CI have their drawbacks in general. For EP+merge, despite that it may be inferior in obtaining outstanding solutions, we notice that the it requires much more MDL metric evaluations than both HEP and EP+CI. The reason is obvious as the absence of the hybrid framework renders EP+merge to search in the entire search space. Consequently, EP+merge will sample much more different solutions and hence a much larger number of metric evaluations. Although the consequence is not obvious in the current data set, it makes a difference when the size of the data set grows. Because it takes $O(n_r)$ time, where n_r is the number of records in the data set, to evaluate the MDL score of a node, it follows that time needed for one evaluation increases with increasing data

set size. When the size of the data set is huge, we expect that the contribution due to the hybrid framework will manifest while EP+merge will run much slower.

For the EP+CI implementation, we notice that the best-so-far solution will often appear later than HEP in terms of the number of generations (ANG). Apart from the difference in mean, the Kruskal-Wallis statistics also suggest there is a significant difference between EP+CI and HEP (p-value equals 0.00025). Hence, in general, within a fixed number of generations, HEP can obtain a better final solution than EP+CI. From our experiment, it is evident that both algorithms are able to obtain similar final solutions in the time span of 5,000 generations. However, in another set of experiments, where we set the termination criterion to be 1,000 generations, HEP easily outperforms EP+CI. Another drawback of EP+CI relates to the reliance on the hybrid framework. If ever the CI tests were not useful, the performance of EP+CI would be greatly affected. Although HEP could also suffer from the same problem, the use of the merge operator will provide a bottom line performance boost as manifested in EP+merge.

With the drawbacks of EP+CI and EP+merge mentioned, a combination of both strategies seems justified.

4.4.2 Effect of Different Population Sizes

In this experiment, we test HEP with three different population sizes (m): 40, 50, and 60. For each setting, we adjust the tournament size accordingly so that the selection pressure is similar. In particular, the tournament size is six for the population size of 40, seven for the population size of 50 and eight for the population size of 60. We summarize our results in Table 4.14.

	AFS	AIS	AET	ANG	AME	ASD
$m = 40$	138,575.8 (481.9)	183,939.3 (6,154.5)	303.8 (20.7)	1,162.9 (1,048.9)	5,042.6 (2,667.9)	6.8 (4.1)
$m = 50$	138,693.4 (538.1)	182,363.7 (5,856.5)	344.2 (23.7)	1,179.3 (1,250.1)	5,292.8 (3,032.8)	9.5 (5.7)
$m = 60$	138,611.2 (499.9)	181,561.7 (5,638.1)	387.7 (27.7)	1,161.0 (1,397.8)	5,640.3 (3,371.1)	8.9 (5.6)

Table 4.14: Performance of HEP with different population sizes.

A large population means that we use more search points. Hence, as reflected

by the average MDL metric evaluation statistics (AME), there are more different solutions examined than in a smaller population. Consequently, the time consumed (AET) also increases proportionally with increasing population size. Although we expect that a larger population would help to obtain the solution earlier, there is no obvious trace in our findings. If we compare the average number of generations needed to obtain the best-so-far solution, we have similar figures (both mean and standard deviations) for different population sizes. Moreover, the statistic obtained in the Kruskal-Wallis test (p-value equals 0.555) also suggests that the difference is not significant. Although a better average score is obtained for m equals 40, we note that this observation misleading. To have a close examination on their performance, we present a more detailed comparison in Table 4.15.

	Best Score	Lower quartile	Worst score	Average
$m = 40$	138,275 (24)	138,668	140,267	138,575.8
$m = 50$	138,275 (17)	138,692	140,265	138,693.4
$m = 60$	138,275 (19)	138,668	140,328	183,611.2

Table 4.15: A comparison of the final scores obtained for different population sizes.

From the table, we observe that the performance of HEP is almost alike for different population sizes. Essentially, most of the runs (out of forty) could return the recorded best score while similar results is obtained in the worst run. In conclusion, since the experimental results do not favor using a larger population, we recommend that a moderate choice of the population size (i.e. $m = 50$) would be sufficient for HEP.

4.4.3 Effect of Different Δ_α

In our implementation, the parameter Δ_α corresponds to the difference of the value of α between a parent and its offspring, if there are any. In this experiment, we investigate the effect of varying the value of Δ_α. We perform testing using the values of 0.0, 0.005, 0.01, 0.02, 0.05, and 0.1. In analyzing the result, we obtain some interesting observations that we did not anticipate. Before we go into details, we present our results in Table 4.16.

From the table, we notice that different runs give largely similar final results (AFS). Furthermore, there are no apparent differences in the time used (AET).

	AFS	AET	ANG	AME	ASD
$\Delta_\alpha = 0.0$	138,765.3 (663.2)	341.1 (10.9)	762.6 (834.2)	3,735.9 (517.2)	8.3 (5.4)
$\Delta_\alpha = 0.005$	138,688.5 (490.8)	335.8 (13.5)	1,000.4 (991.3)	3,805.4 (679.5)	9.9 (5.2)
$\Delta_\alpha = 0.01$	138,830.0 (569.8)	343.9 (15.3)	1,161.3 (1,226.5)	4,342.1 (1,449.4)	10.0 (5.3)
$\Delta_\alpha = 0.02$	138,693.4 (538.1)	344.2 (23.7)	1,179.3 (1,250.1)	5,292.8 (3,032.8)	9.5 (5.7)
$\Delta_\alpha = 0.05$	138,642.6 (454.4)	344.1 (33.7)	1,119.1 (1,019.6)	5,198.6 (3,702.8)	8.7 (4.8)
$\Delta_\alpha = 0.1$	138,672.5 (504.8)	331.5 (14.0)	818.7 (943.2)	3,782.6 (1,579.6)	9.4 (5.1)

Table 4.16: Performance of HEP with different Δ_α.

Nevertheless, both the ANG and the AME measures show an similar and interesting trend upon different values of Δ_α: they peak at a moderate value of Δ_α (i.e. $\Delta_\alpha = 0.02$) and diminish with both increasing or decreasing values of Δ_α.

To account for this observation, we first note that α has a "freezing" side effect. Recalling that smaller α implies a more restricted search space, in the extreme case that $\alpha = 0$, any edge addition to the network is forbidden. When this happens, most of the mutation operations on the individual will left the structure intact. With this freezing possibility, a superior individual is likely to be *driven* or *attracted* towards having $\alpha = 0$ as a consequence of the selection pressure.

To provide a more intuitive illustration, suppose that the best-so-far solution, G_p, appears at a certain generation and its α equals α_p. In the next generation, the individual will produce an offspring (1) G_+ that has α equals $\alpha_p + \Delta_\alpha$ or (2) G_- with $\alpha_p - \Delta_\alpha$ or (3) the one with α_p. To put forward our argument, we simply ignore the last case for it only begins another cycle in the next generation if the value of α_p does not change for the current generation. For G_+ and G_-, if we compare their chance of survival, it can be concluded that G_- will be the better one to survive. The reason is because G_- will have a smaller chance to commit a mistake during mutations. Although that both offspring will have an equal chance to remove an edge erroneously, $G-$ will have a comparatively smaller chance to add an improper edge (as a smaller α implies more constraints). Hence, the evolutionary process would favor a decrement. It follows that, eventually, the best-so-far solution will have α equals zero. In the experiment, we obtain similar observations favoring the argument. In Table 4.17, we present the number of trials (out of forty) that the

Setting	Counts
$\Delta_\alpha = 0.0$	0
$\Delta_\alpha = 0.005$	14
$\Delta_\alpha = 0.01$	19
$\Delta_\alpha = 0.02$	25
$\Delta_\alpha = 0.05$	30
$\Delta_\alpha = 0.1$	36

Table 4.17: Counts of zeroes.

best-so-far solution has $\alpha = 0$ ultimately.

Evidently, with a larger Δ_α, there is higher chance for α of the best-so-far solution to become zero. Because "the fittest the survival", gradually, the entire population will incline towards having α equals zero. By then, every individual will tend to "freeze" its network structure which means that there will be little further exploration in the search space. As a result, the search process will be less likely to escape from a local optimal point and virtually ceases to improve. Since a larger Δ_α also implies a faster convergence (decrement by Δ_α to "freezing"), this explains the diminishing trend of ANG and AME when Δ_α becomes bigger. Although one could argue that a larger Δ_α could help to escape (increment by Δ_α) from freezing more easily, it also means a higher chance of adding the *wrong* edge because the *wrong* edge may have a higher p-value. From the result shown in Table 4.16, it seems that the balance is obtained when Δ_α takes the values of 0.02 and 0.05 in that they sample much more candidate solutions before convergence.

To explain for the diminishing trend at the other end, namely when Δ_α is decreasing, we note that the result is due to a different kind of "freezing". Suppose, again, that the best-so-far solution appears in a certain generation and its α equals α_p. In the next generation, its offspring will inherit the value in the range of $\alpha_p \pm \Delta_\alpha$. When Δ_α is small, the range will be small which possibly does not make any difference to the search space. Hence, the chance of survival for α going up or going down is equally likely. Although the entire population will slowly be alike to the best-so-far solution, it is expected that α of every individual will not be differ much from one another. In other words, the cutoff value of the entire population is likely to be bounded in a certain range: $\alpha_p \pm k\Delta_\alpha$ where k denotes the finite steps that the population takes to converge. As a consequence, the searching will be localized in a certain area within the entire search space which constitutes another kind of "freez-

ing". This explains the diminishing trend of ANG and AME when Δ_α becomes smaller.

Although Δ_α affects the search pattern, there seems to be little impact on the ultimate result. Possibly, the data set in our experiment does not manifest a problem so that different search patterns would approach similar solutions towards the end. However, given that the overhead (i.e. time used) is similar for different Δ_α, we would recommend a moderate choice of Δ_α (i.e. 0.02 or 0.05). The reason is that when Δ_α takes a moderate value, it exhibits a more active and lively searching process (less "freezing") which would possibly help to escape from local optimal for certain problem instances.

4.4.4 Efficiency of the Merge Operator

In this section, we study the number of successful merge operations during a run. In our current implementation, we designate a fixed number of individual (half of the population) to undergo merging. When merging fails to produce a better individual, the result is discarded which would mean a waste of effort. Hence, it is important to note the success rate as this indicates the effectiveness of the merge operator. In Figure 4.7, we plot the count of successful merge operations in each generation (sampling at every ten generations) of a single run.

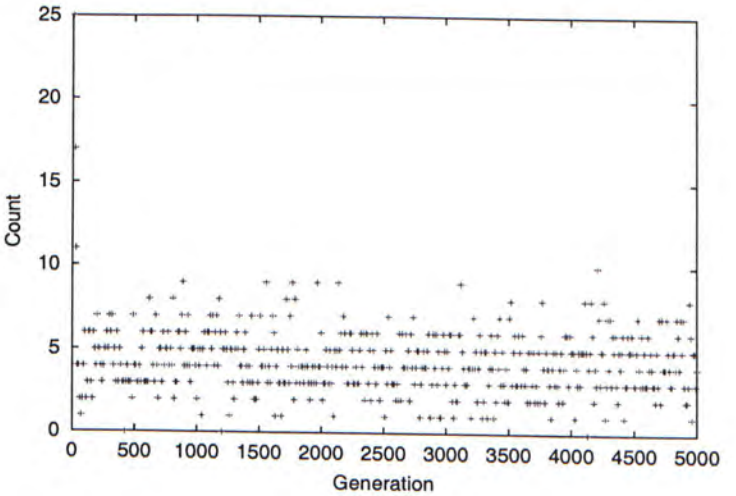


Figure 4.7: Successful merging in a run.

As shown in the figure, the number of successful cases fluctuates between zero and ten. This is, nevertheless, a good sign as the success rate is almost constant which implies that the efficiency of the merge operator does not decline over a period of time. If we look at the statistics, we obtain an average figure of 4.4 and the success rate is therefore nearly one-fifth (= 4.4/25). Although we do not have a standard to judge whether a successful rate of one-fifth is good, it probably suggests a good starting point for improvement: to adjust the current scheme so that the successful rate could be made higher.

4.5 Summary

In this chapter, we present various experimental results on comparing the performance of different learning algorithms and on studying and analyzing CCGA and HEP. In comparing the performance of our proposed algorithms against MDLEP, it is evident that our algorithms are more efficient than MDLEP. Most importantly, both algorithms executes significantly faster than MDLEP which is an advantage in practice. When making a comparison between CCGA and HEP, we argue that both algorithms could do well in different aspects but neither could dominate.

In studying CCGA, we evaluate the performance of the algorithm under different settings. As suggested by our findings, the hybrid framework and the choice of α play an important role in CCGA. If the choice is improper, the performance of CCGA will be affected significantly. Concerning the choice of the parameter for CCGA, we recommend a smaller population size (i.e. 20), a low mutation probability (i.e. 0.05) and a moderate amount of crossover probability (i.e. 0.7). Finally, we investigate the effect of using different belief factor and find that using a value of 0.2 seems to give a promising performance.

We analyze HEP in a similar way. Since both the hybrid framework and the merge operator helps to improve the efficiency, we investigate the performance of HEP without them. Although our findings is not conclusive of a particular recommendation, we note the merits of bringing both strategies together which justifies the HEP framework. Concerning the choice of parameters, we recommend using moderate values for the population size (i.e. 50) and $\Delta\alpha$ (i.e. 0.02). In collecting

the statistics of successful merging, we obtain an approximated figure of one-fifth success rate. Clearly, if we aim to improve HEP further, we could try to improve the success rate as each failed merge operation is a waste of effort.

Chapter 5

Learning Bayesian Network Classifiers

Using the result from Bayesian network learning, we extend our work to learning Bayesian network classifiers. Although the objectives of the learning problems are different, we observe similarities between the learning algorithms for which our work could be applied. In particular, we target on learning two special types of Bayesian network classifiers: the augmented network classifier and the multinet classifier. Although the two classifiers have also been discussed and evaluated in [26], we suspect that the previous study may employ greedy heuristics in construction, which could learn inferior models. In contrast, by using evolutionary computation techniques, we expect better models are obtained so that a fair comparison can be made.

We choose HEP to be used in our study. Because both of our algorithms seem to deliver similar performance, we expect that the experimental result would be similar for the two algorithms. However, one reason that favors our choice is that HEP could avoid the problem of selecting an improper cutoff value. Concerning the parameter values, the default setting as shown in the previous chapter is adopted for simplicity.

We organize this chapter as follows. In Section 5.1, we discuss the problems of a straightforward approach to learn a Bayesian network classifier. Due to the problems mentioned, we put forward the two special classes of Bayesian network classifiers, the multinet classifiers and the augmented network classifiers in Section 5.2 and

Section 5.3 respectively. In Section 5.4, we describe the method that we employed in assessing the performance of the proposed classifiers. In Section 5.5, we present the results and various comparisons. In Section 5.6, we discuss the experimental result and the insights that we gained. In Section 5.7, we study a novel application by which Bayesian network classifiers are used for response model construction in the direct marketing problem. Finally, we summarize our findings in Section 5.8.

5.1 Issues in Learning Bayesian Network Classifiers

Let $\{A_1, \ldots, A_n\}$ denotes the set of attributes and let C denotes the class variable. In the simplest case, we can apply any Bayesian network learning algorithm on the training set, which consists of $\{A_1, \ldots, A_n, C\}$, and use the network returned as a classifier. We call this kind of classifier the general Bayesian network classifier (GBN). However, a GBN can perform poorly in practice [26]. To understand the reason, it is important to note that learning a *good* Bayesian network is quite different from learning a *good* classifier.

In the classification problem, we aim at finding a classifier with the best distinguishing power. Referring to equation 2.9, the best classifier should be the one that faithfully represent $P(C \mid A_1, \ldots, A_n)$. However, for general Bayesian network learning algorithm, the objective is to find a good approximation to the joint probability distribution $P(C, A_1, \ldots, A_n)$. Clearly, the two objectives are different by nature. In a simpler sense, a GBN fails to be a good classifier because during learning, it does not distinguish between the class variable with the attributes. If the MDL metric is in use, the performance of GBN will be aggravated further for problems that contain many class values [26]. In particular, it incurs a heavy cost when an edge is added between the class node and any attribute node. This implies a reluctance to admit a possible influence of the attribute value on the class value. Consequently, the resultant network would give poor performance on classification.

Such observation signifies that it is necessary to speculate how to learn a good Bayesian network classifier using a general Bayesian network learning algorithm.

Since we adopt a *search* approach, the problem of a general Bayesian network learning algorithm is that it is not searching for a good Bayesian network classifier. A key which determines the objective of a search algorithm is the scoring function. Hence, one remedy is to employ other scoring functions (other than MDL) that could evaluate a network for being a classifier. If such metric exists, it will be straightforward to modify our network learning algorithms into a classifier learning algorithms by substituting the MDL metric with the new metric. Unfortunately, as discussed in [26], such scoring metric is difficult to compute in practice.

As we cannot directly apply our algorithms on classifier learning, the strategy is, therefore, to apply our learning algorithms to search for what approximates a good classifier. In the following sections, we describe two approaches that fulfill the objective.

5.2 The Multinet Classifier

Although a Bayesian network can be used to approximate a joint probability distribution using independency relations, there is an inherent problem, namely that the Bayesian network fails to capture context-specific independence [42, 71]. Implicitly though, we assume that the same independence assertions (that is depicted in the network) hold in every observations. However, this assumption can be violated. In particular, it is possible that certain independence assertions are valid only under a particular circumstance (context) while a different set of assertions is valid in another circumstance.

It turns out that it is the deficiency [29] of the Bayesian network to represent context-specific independence. Because of this, Geiger and Heckerman [29] seek for more powerful representations using multiple networks. Although their work has a detailed description of the structures and the inference algorithms, it does not mention the learning algorithm as they assume that the networks are built by consulting with experts. In a later article [71], Heckerman et al. fill this gap by proposing a method to learn the multiple networks from data. Following [71], the multiple network representation is called a Bayesian multinet. In essence, the Bayesian multinet comprises of a set of *component Bayesian networks*, each of which

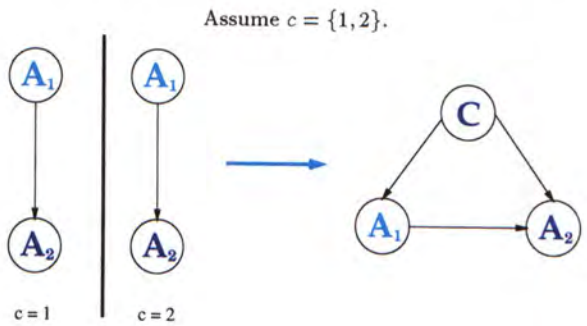


Figure 5.1: Equivalence between the multinet and a single network representation.

encodes the probability distribution of a particular state of a distinguished *context* variable.

By modeling the class variable, $C = \{c_1, \ldots, c_k\}$, as the distinguished context variable, we can extend the multinet approach to the classification problem. We call such classifier a Bayesian multinet classifier (MN). In particular, the multinet classifier is a set of component networks $\{G_1, \ldots, G_k\}$, where each G_i encodes the distribution $P(A_1, \ldots, A_n)$ for a particular class value c_i (i.e. $P(A_1, \ldots, A_n \mid C = c_i)$). A new instance $I = \{a_1, \ldots, a_n\}$ is labeled c_i if for $j = 1, \ldots, k$,

$$P_{G_i}(a_1, \ldots, a_n)P(C = c_i) \geq P_{G_j}(a_1, \ldots, a_n)P(C = c_j) \tag{5.1}$$

where P_{G_i} denotes the probability of the occurrence of the instance evaluated from the component network G_i.

Intuitively, at the expense of using more parameters to describe a distribution, we expect that a multinet approach will lead to an improved performance. Following the assumption, if the attributes $\{A_1, \ldots, A_n\}$ exhibit different independency relations for different class values, a Bayesian multinet classifier will have a clear advantage over a single network classifier. In case that this assumption is inappropriate and that every component network exhibit the same independency relations, the multinet representation merely equals a single network if we assume that there is no redundant attributes (i.e. there is an edges going from the class node to every attribute node in a single network). We picture the idea in Figure 5.1.

For practical reasons, the multinet approach is also a preferred approach as we could easily extend our Bayesian network learning algorithm to learn a multinet classifier. Similar to the general Bayesian network learning problem, learning a

component Bayesian network also has the same goal, which is to find the best representation of the joint distribution $P(A_1, \ldots, A_n \mid C)$. Thus, a Bayesian multinet classifier is readily built by applying our algorithm to learn each component Bayesian network individually. We show the procedures in Figure 5.2.

- Partition the training set D into disjoint subsets $\{D_1, \ldots, D_k\}$ according to the class label $C = \{C_1, \ldots, C_k\}$ of each instance.
- For $i = 1, \ldots, k$,
  - Discard the class information in the data set D_i.
  - Learn a component Bayesian network G_i (using HEP) from D_i consisting of $\{A_1, \ldots, A_n\}$.
- Return the networks $G_1, \ldots, G_k$ as the multinet classifier.

Figure 5.2: The multinet classifier learning algorithm.

We note that the Bayesian multinet classifier is not a new idea. Interestingly, in more than three decades ago, Chow and Liu have already used a multi-tree (component networks are trees) approach for classification [18]. Recently, Friedman et al. [26] have reported experimental results of a Bayesian multinet classifier. However, since they did not elaborate on the detail of the learning algorithm, we suspect that they use a greedy approach which may result in sub-optimal solutions. Although we did not include the comparison in this dissertation, our claim seems to be justified as the performance of our multinet classifier shows an improvement over the reported result. However, it must be mentioned that the deviation may as well be due to different experimental settings.

5.3 The Augmented Bayesian Network Classifier

As argued above, a general Bayesian network usually has poor classification performance. One reason is that the learning procedure does not distinguish the class node from other attributes. To this, we can adopt the augmented network approach [26] so that by biasing the network structure, we put an emphasis on the class node. In other words, we mandate the class node to be the parent of every attribute node so

Procedure MakeNaïVe(G)

- For every attribute A_i $(i = 1, \ldots, n)$,
  - Remove $A_i \rightarrow C$, if existed.
  - Check if $Pv_{CA_i} \leq \alpha_{naive}$,
  - if so, add $C \rightarrow A_i$.

Figure 5.3: The `MakeNaïve()` procedure.

that the classifier possesses a naïve Bayesian classifier structure as a basis. We call the classifier that having such biased structure the augmented Bayesian network classifier (ABN). As discussed before, the state-of-the-art Bayesian network classifier, the tree-augmented Bayesian network classifier(TAN), is also an augmented network classifier, in which the augmenting edges form a spanning tree connecting every attribute node.

Despite that TAN performs outstandingly in practice, it is natural enough to question whether an unrestricted augmenting structure could lead to a further improvement. In other words, is there performance gain if we do not presume a tree structure, and allow a general structure which could be simpler or more complex?

Targeting on this, we attempt to investigate the performance of an augmented network classifier with a general structure. Although the same idea has also been examined by Friedman et al. [26], they have not explicitly mentioned the way to learn the augmented network. We suppose that they use a greedy search approach, which is, again, possible to yield an inferior model. In contrast, we propose an evolutionary search approach that is generally believed to outperform a greed search. Furthermore, our approach distinguishes from theirs in that the basic structure we assumed *may* differ from a naïve Bayesian network structure. In other words, it is not necessary that every attribute node will have the class node as its parent. To implement this idea, the learning algorithm is equivalent to the HEP learning algorithm but with the addition of a `MakeNaïve()` procedure which is executed whenever an individual is created.

Essentially, the `MakeNaïve()` procedure puts the stated bias through adding the *essential* edges. However, not all $C \rightarrow A_i$ are added. In particular, we determine

whether the edge $C \rightarrow A_i$ should be added by checking against Pv_{CA_i}, which is the maximum p-value of the conditional tests between C and A_i (confined to order-0 and all order-1 test) stored in Pv during the CI test phase (see Section 3.4.1). If the p-value is larger than a fixed threshold α_{naive}, we assume that C and A_i are conditionally independent and the edge $C \rightarrow A_i$ is not added.

"So," one would ask, "why is it better to add only selected ones but not all the *necessary* edges?" Although it is true that we intend to learn a network with, essentially, a biased structure, we note that, if we are to add all $C \rightarrow A_i$, we assume that the class is dependent on every attribute node. In some case, where an attribute is spurious or redundant, this assumption will be misleading. While there is statistics suggesting the conditional independence between A_i and C, it will be reasonable to accept the finding and reject the possibly misleading assumption. Thus, we choose to add only those *essential* out of the *necessary*. We note that a similar idea is also employed in [23] for learning a Bayesian network classifier.

Nevertheless, our current approach brings another problem, namely, to choose the value of α_{naive}. Indeed, is it necessary to introduce a new parameter? And, is it possible to use α_i (the cutoff value associated with each individual) in place of α_{naive}? Although this idea seems plausible, we consider that this would violate our aim. If we do not assume a fixed α_{naive} and allow the evolutionary process to select a proper one, it will ultimately be equivalent to searching a general Bayesian network classifier as it is the MDL score which dictates. In other words, it does not evaluate on different biased structures (due to different α_{naive}) but only concerns how good the entire structure approximates the distribution. On the other hand, if we fix the value of α_{naive}, the MDL score helps to reflect how good different augmenting structure is (for a particular biased structure). Evidently, this is more desirable and explains our current approach in using a fixed α_{naive}. Although it is unavoidable that we pick an improper value of α_{naive}, we try to minimize the risk by using a moderate value (i.e. 0.5) for all our experiments.

5.4 Experimental Methodology

We have implemented our classifier atop of the $\mathcal{MLC}$++ system [41], which is a well-known software package targeting on the classification problem with a standard application programming interface (API). $\mathcal{MLC}$++ is short for a "**M**achine **L**earning library implemented in the **C++** programming language." $\mathcal{MLC}$++ provides an ideal workbench for developing new classification algorithm as it comes with the implementation of many standard procedures, like discretization, feature selection and accuracy estimation. Furthermore, it also provides the implementation of many classical classification algorithms, like the naïve Bayesian classifier, the nearest neighbor classifier etc. Altogether, this facilitates performance comparison and ensures that our experiment is conducted in a fair and formal manner with result being reproducible. Also, since some previous research also use $\mathcal{MLC}$++ [16,26], we simply follow the practice to use $\mathcal{MLC}$++ for developing our algorithm and for performance comparison.

We have selected 19 machine learning data sets appearing in the UCI repository [9] which are obtained from the $\mathcal{MLC}$++ web site. The data sets that we obtained are different from the original source in that they are formatted specifically for use with $\mathcal{MLC}$++. Furthermore, each data set includes, in addition to an entire data set, a default training set and a default testing set, which facilitates performance comparison. Besides the machine learning data sets, we include an artificial data set called **asia*3** primarily for evaluating the multi-net classifier. The **asia*3** data set is generated from three Bayesian network. As shown in Figure 5.4, the first network is the original ASIA network, the second one is an empty network and the third one is a random structure. For the last two networks, the parameters are randomly initialized. Assuming that each network corresponds to the dependency relations persists for a particular class value, we simulate 1,000 cases for each network and mix them together. For performance evaluation, we randomly partition the 3,000 case data set, in which 2,500 cases is used as the training set and the rest as the testing set. In Table 5.1, we show the list of data sets that we have used in our experiment. For each data set, we also show the size, the number of attributes, the number of class values, and the accuracy estimation method. As a

general guideline for selecting the accuracy estimation method, we will use ten-fold cross validation on the entire data set when the data set is small [39]; and when the data set is large, we will simply use the default training set for learning and the default testing set for performance evaluation.

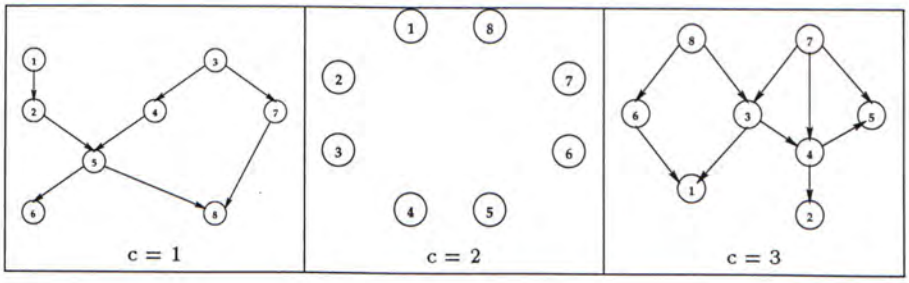


Figure 5.4: The multi-net that generates the artificial data set.

	Data set	Attributes	Classes	Data set sizes	
				Training set	Testing set
1	adult	13	2	32561	16281
2	asia*3	8	3	2500	500
3	australian	14	2	690	CV
4	breast	10	2	683	CV
5	cars	9	3	392	CV
6	chess	36	2	2130	1066
7	crx	15	2	653	CV
8	diabetes	8	2	768	CV
9	flare	10	2	1066	CV
10	german	20	2	1000	CV
11	glass	9	7	214	CV
12	glass2	9	2	163	CV
13	heart	13	2	270	CV
14	iris	4	3	150	CV
15	letter	16	26	15000	5000
16	pima	8	2	768	CV
17	solar	12	6	323	CV
18	splice1	62	3	2125	CV
19	vehicle	18	4	846	CV
20	vote	16	2	435	CV

Table 5.1: Data sets used in the experiments.

Currently, we do not deal with missing data. Hence, an instance with missing value is simply discarded. Since our Bayesian network classifiers can only handle discrete attributes, we perform a pre-discretization step on the data set if continuous attributes are present. Inside $\mathcal{MLC}$++, there are a number of discretization methods provided. With no particular preference, we use the entropy discretizor, which is the default discretization method. For experiments running cross-validation, we

$\mathcal{MLC}$++ options	Values
PERF_ESTIMATOR	cv or test-set
CV_FOLDS	10 (default)
CV_TIMES	1 (default)
PERF_EST_SEED	7258789 (default)
LOGLEVEL	2

Table 5.2: MLC++ configurations used in the experiments.

note that the partitioning of the data set is the same for each run. In other words, the training set and the testing set is the same for the i-th fold in testing various classification algorithms. For reproducibility purpose, we summarize the $\mathcal{MLC}$++ option values that we adopted in Table 5.2.

In choosing the data set used in the experiments, we have some general criteria to help making the decision. First, we would avoid small data sets. For small data set, we refer to the data set that has less than 200 instances. In general, since our classifier model often involves many parameters (i.e. CPTs at each node), estimation of the parameters may not be accurate if a small data set is in use. Furthermore, a larger data set helps to make a fair comparison by mitigating the effect of "smoothing" in TAN. In their original article [26], Friedman et al. found that when they use a technique called smoothing in parameter estimation, the prediction accuracy of the classifier is improved. Despite that smoothing seems to be a reasonable approach, it is not fully understood that this approximation can help [1]. In other words, was the better performance a merit of the classifier learning algorithm or was it the merit of a better parameter estimation method? Suffice it to say, the effect of smoothing will be diminished, in general, when a larger data set is in use. Hence, to make a more fair comparison between TAN and our "un-smoothed" classifiers, we prefer to use a larger data set.

Second, because we do not deal with missing values in a data set, we would not use data sets that have many missing values. In the extreme case where a training set or a testing set is empty because every instances containing missing values are discarded, the experimental result is simply fallible.

[1] The smoothing technique is attacked by [12] claiming the approach to be "not understandable from a theoretical point of view."

Third, we would opt for data sets with which the entropy discretizor performs effectively. In certain data sets, the discretizor can produce *strange* result such that a discretized attribute consists of only one value, or, in another extreme, consists of numerous values. If an attribute only consists of a single value, it will surely be redundant as its presence or absence will not carry any significance. If an attribute contains numerous values, including the node as a parent will incur a heavy cost as evaluated by the MDL metric. In order to avoid such *strange* and possibly misleading result, it is important to check whether the result of discretization is reasonable.

5.5 Experimental Results

For comparison purpose, we have implemented the TAN classifier in $\mathcal{MLC}$++ (with smoothing). In addition, we have obtained an implementation of the well-known decision tree classifier, C4.5 [62]. In Table 5.3, we report the performance of different classifiers evaluated on the twenty data sets. We use the following abbreviations:

NB The naïve Bayesian classifier provided with the $\mathcal{MLC}$++ package.

GBN A general Bayesian network classifier which is learned using HEP.

ABN The augmented Bayesian network classifier.

MN The Bayesian multinet classifier.

TAN The Tree-augmented Bayesian network classifier.

C4.5 The well-known decision tree classifier.

The figures at each entry shows the average predictive accuracy as reported by $\mathcal{MLC}$++. The number appearing on the right of the $\pm$ shows the standard deviation of the accuracy [40]. [2] For the **cars** data set, the C4.5 program fails due to unknown reason, hence, the corresponding entry is omitted. To provide a rough comparison among the classifiers, we also compute the average of the predictive accuracy over the twenty data sets. In this regard, TAN would seem to be the best classifier, closely followed by ABN and MN; while the worst classifiers are the naïve Bayesian classifier and GBN.

[2] If only the testing set is used, the standard deviation is calculated assuming the classification process is a Bernoulli trial. If cross validation is used, the standard deviation shows the standard deviation of the sample mean. [39]

Data set	NB	GBN	ABN	MN	TAN	C4.5
adult	83.84 ± 0.30	84.59 ± 0.29	85.56 ± 0.29	85.21 ± 0.29	85.47 ± 0.29	86.28 ± 0.28
asia3	84.40 ± 1.62	90.60 ± 1.31	90.60 ± 1.31	91.80 ± 1.23	88.80 ± 1.41	91.20 ± 1.27
australian	85.65 ± 1.44	85.80 ± 1.35	86.81 ± 1.41	86.67 ± 1.45	85.07 ± 0.99	85.36 ± 1.25
breast	97.37 ± 0.47	97.37 ± 0.65	96.93 ± 0.59	96.93 ± 0.67	97.22 ± 0.74	95.47 ± 0.83
cars	98.22 ± 0.55	97.71 ± 0.46	97.46 ± 0.53	98.22 ± 0.55	95.15 ± 1.11	N/A
chess	87.15 ± 1.03	93.53 ± 0.75	94.56 ± 0.70	96.53 ± 0.56	92.12 ± 0.83	99.53 ± 0.21
crx	85.61 ± 0.89	85.00 ± 0.37	86.83 ± 0.77	85.45 ± 0.70	85.62 ± 1.29	86.68 ± 1.06
diabetes	75.13 ± 1.15	74.09 ± 1.15	75.39 ± 1.07	76.17 ± 1.17	76.17 ± 1.01	74.47 ± 1.46
flare	79.46 ± 1.57	82.92 ± 1.55	81.89 ± 1.56	80.77 ± 1.28	82.54 ± 1.52	82.26 ± 1.31
german	74.10 ± 1.80	69.70 ± 1.24	74.60 ± 1.40	73.30 ± 2.18	72.70 ± 1.52	71.20 ± 1.29
glass	71.47 ± 2.15	62.73 ± 3.88	71.47 ± 2.15	64.55 ± 2.89	70.13 ± 2.37	71.00 ± 2.64
glass2	80.33 ± 2.00	70.51 ± 2.46	81.54 ± 1.90	79.12 ± 2.09	80.92 ± 3.04	79.04 ± 2.70
heart	82.22 ± 1.73	81.11 ± 2.79	82.96 ± 1.76	81.85 ± 1.87	82.59 ± 1.84	82.22 ± 3.21
iris	92.67 ± 1.85	94.67 ± 1.94	92.67 ± 1.85	92.67 ± 1.85	93.33 ± 1.72	95.33 ± 1.42
letter	74.94 ± 0.61	74.98 ± 0.61	76.56 ± 0.60	79.84 ± 0.57	85.98 ± 0.49	77.70 ± 0.59
pima	75.90 ± 1.75	75.64 ± 1.82	75.12 ± 1.43	75.65 ± 1.54	75.77 ± 1.57	74.99 ± 1.74
solar	71.21 ± 1.46	74.01 ± 1.38	71.24 ± 1.45	73.70 ± 1.83	72.13 ± 1.40	71.22 ± 1.82
splice1	94.64 ± 0.69	95.39 ± 0.64	94.74 ± 0.68	94.64 ± 0.69	94.36 ± 0.71	93.33 ± 0.77
vehicle	61.12 ± 1.72	60.64 ± 1.85	69.73 ± 1.70	67.25 ± 1.36	71.28 ± 2.01	69.14 ± 1.65
vote	90.34 ± 0.68	93.80 ± 0.59	91.25 ± 0.77	91.27 ± 0.95	93.32 ± 0.88	95.64 ± 0.52
AVERAGE	82.29	82.24	83.90	83.58	84.03	83.27

Table 5.3: A summary of performance of different classifiers.

To conduct a more detailed comparison between two classifiers, we make use of two graphical representation, the error curve and the scatter plot [26]. In an error curve, we plot the average error rate together with the 90% confidence interval [24] for each data set. In case where only the testing set is used for evaluation, the 90% confidence interval is estimated as in [39]. Since the plot indicates *error* rather than *accuracy*, the lower the error, the better the performance. Moreover, we arrange the data set ordering according to the difference in the error rate so that the two curves, representing the two classifiers, cross only once [26]. In a scatter plot, we plot the average error rates (on all the experiments) of the two classifiers against each other. Suppose that the horizontal axis stands for the error rate of our proposed classifier, points situating above the diagonal line will correspond to the cases where our proposed classifier performs better. Hence, if most points situate above the diagonal lines, it indicates that the proposed classifier is better on most data sets. If most points situate near the diagonal line, it indicates that the two classifiers have similar performance.

From the results, we observe that GBN often has a poor performance in compared with other Bayesian network classifiers. Although GBN also excels in a few data

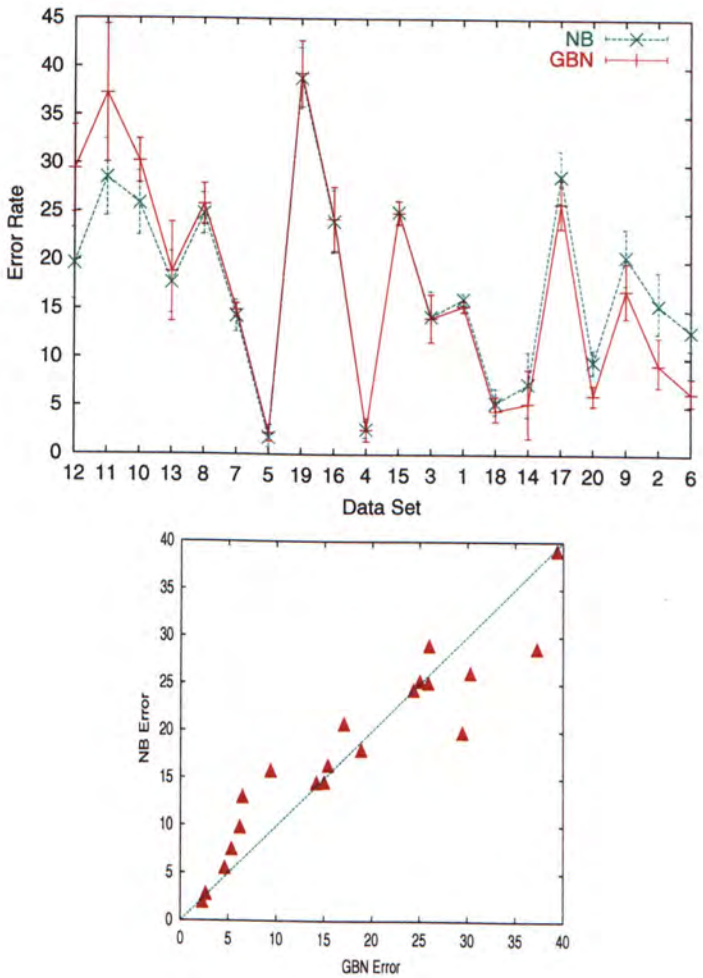


Figure 5.5: Comparing GBN with NB.

sets, it performs exceedingly poor in the **vehicle** and **glass** data set, where the data set is small but the number of classes are many, for reasons that discussed before. In Figure 5.5, we provide a comparison of GBN with the naïve Bayesian classifier. With an uneven and, in most cases, poorer performance, it confirms our expectation that GBN fails to be a good (i.e. robust) classifier.

Although we expect that ABN and MN will have a better performance than TAN because of an unrestricted structure, the difference turns out to be not significant in many data sets. In Figure 5.6, we produce the scatter plot and the error curve comparing ABN with TAN. Apart from the **letter** data set in which TAN significantly outperforms all the other classifiers, ABN performs equally well with TAN

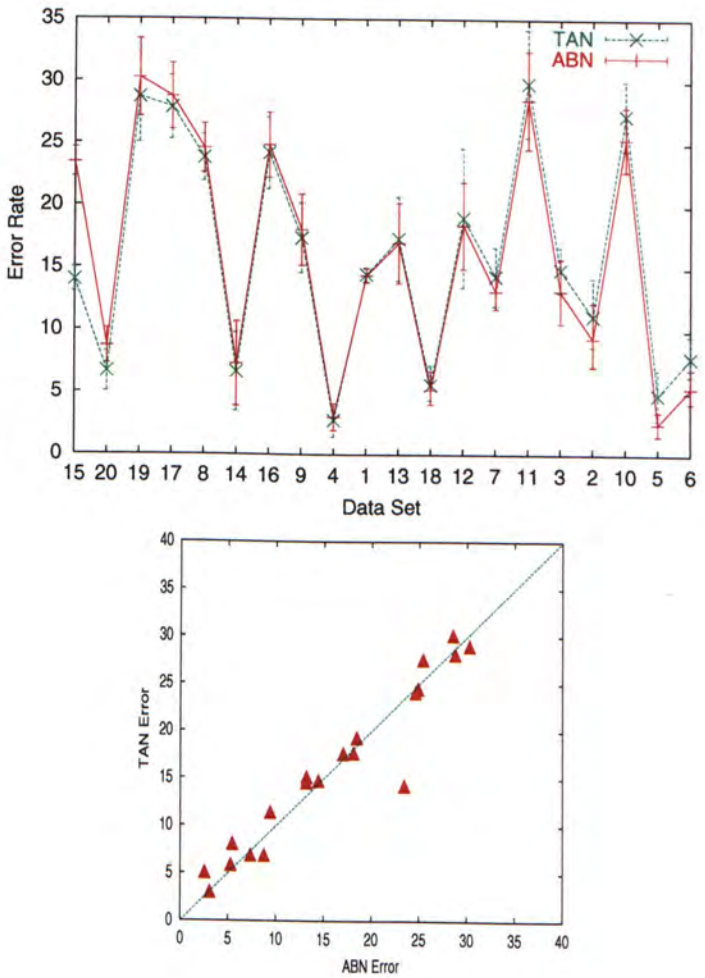


Figure 5.6: Comparing ABN with TAN.

across many data sets. On the other hand, ABN seems to be a clear winner when compared with the naïve Bayesian classifier (Figure 5.7). In general, this suggests that while the presence of an augmented structure leads to an observable difference in classifier performance, the advantage of using an unrestricted structure is not clear. We defer our discussion on this topic in later paragraphs.

In Figure 5.8, we show the comparison between our multinet classifier and TAN in the error curve and the scatter plot representations. Despite that the multinet classifier and TAN have similar performance in a number of data sets, there is a bigger disparity between them. Notably, for the **chess** data set, the multinet classifier has the best performance among all the Bayesian network classifiers. When

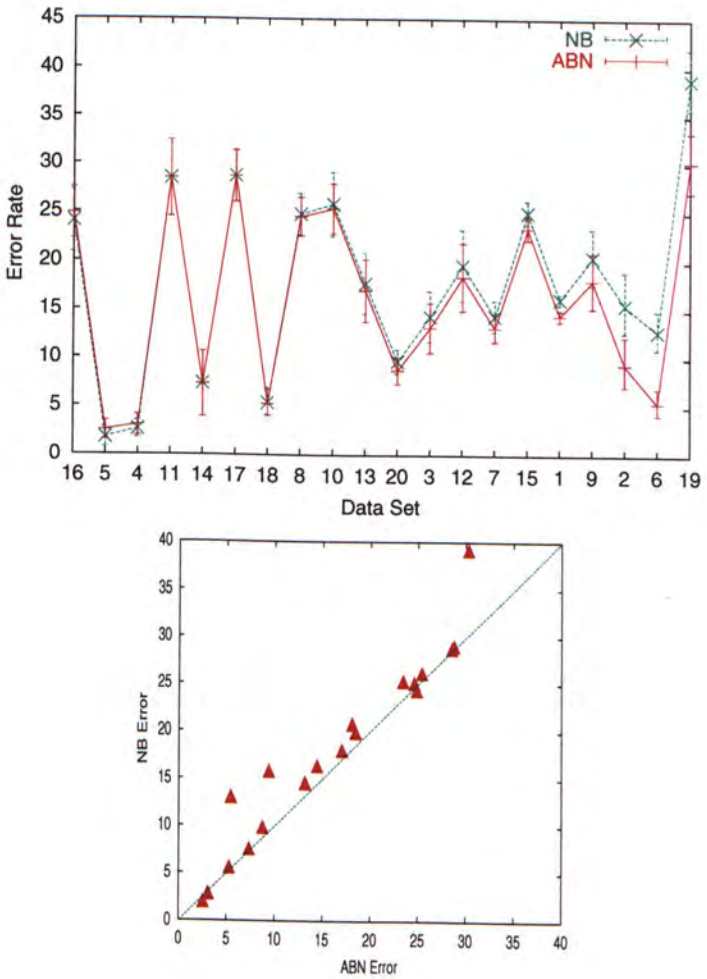


Figure 5.7: Comparing ABN with NB.

we examine the resultant networks learned for the two different classes, we observe that the structural difference between them is large. In particular, the two networks have respectively 57 and 65 edges (the problem has 36 attributes) and the structural difference between them is 72. This demonstrates that for domains exhibiting very different dependency relations for different class values, as in the **chess** data set, the multinet classifier is able to perform very well. On the other hand, the multinet approach performs poorly in the **glass** and the **vehicle** data sets. We conjecture that the reason for this obvious degradation in performance is due to small data sets having many classes. Recalling that a composite network is learned from a partitioned training set, there will be little data contained if the data set is small.

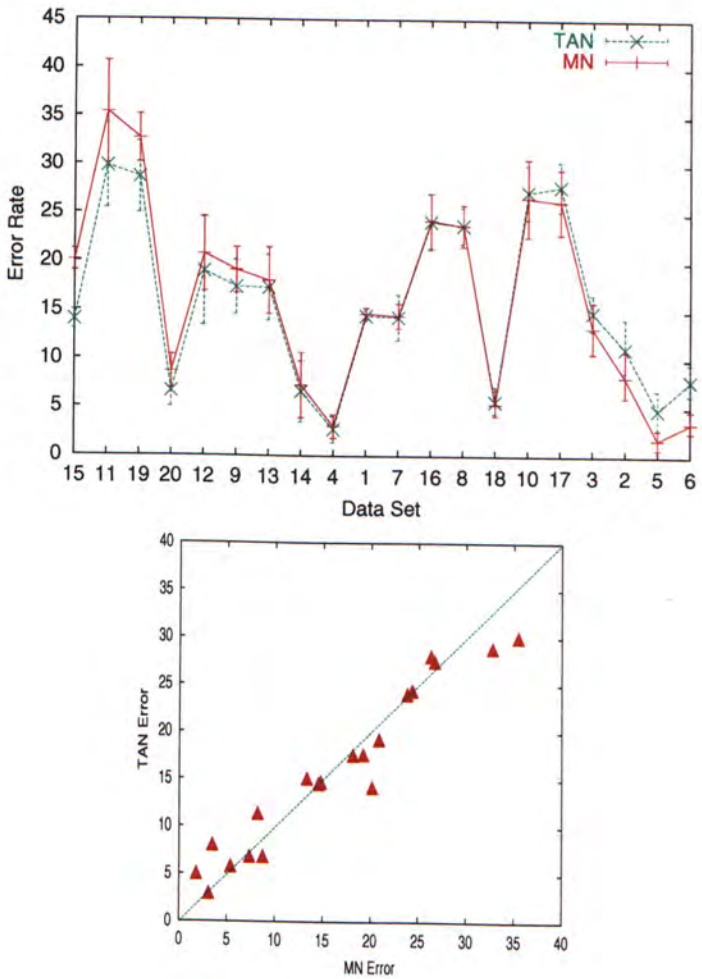


Figure 5.8: Comparing MN with TAN.

The problem is more severe if there are many classes as we need to divided the data set into more partitions. Consequently, a composite network may easily overfit the data. Moreover, it presents a difficulty in estimating the parameters of the networks due to data insufficiency. Take the **glass** data set as an example, it is observed that the smallest subdivided data set only consists of nine instances during one of the ten-folds.

In comparing our multinet classifier with the naïve Bayesian classifier, we observe similarly that our multinet classifier generally outperforms its opponent. Except for the **glass** data set in which the multinet classifier performs much worse, our classifier performs equally well or better for the other data sets.

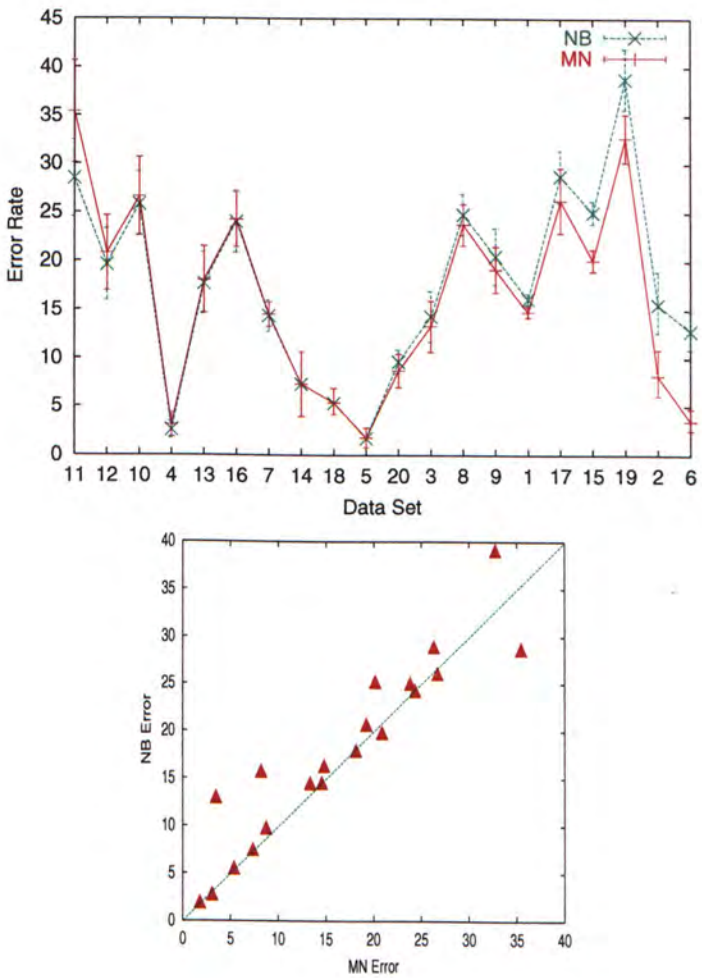


Figure 5.9: Comparing MN with NB.

5.6 Discussion

It surprises us that the proposed classifiers, which attempts to make a better approximation, do not improve much over TAN in practice. For most of the data sets, our proposed classifiers obtain similar performance in compared with TAN. While for a few data sets (e.g. **chess**, **cars**), our classifiers seem to excel, poorer performance are observed for some other data sets (e.g. **letter**, **glass**). So, what are the possible reasons?

It seems that the probability distributions of most real-life data set are not *odd* at all, which implies that a simpler approximation would suffice. From the **chess** data set, we observe that the multinet classifier, by being able to capture

complicated context-specific independence, has superb performance compared with other Bayesian network classifiers. However, for most data sets, there seems to be an absence of the context-specific independence in the sense that the composite networks tend to be simple and do not differ much from one network to another. Hence, is our basic premises, that a domain contains complicated independency relations, unrealistic in practice? And if our supposition is right, a classifier with a more complex model (i.e. ABN and MN), may not provide a significant difference in performance but will run the risk of over-fitting the data.

From another perspective, even that underlying distribution is *odd*, different classifiers can still perform well as evident from the artificial data set, **asia*3**. Although the data set is manipulated in such a way that it favors the multinet classifier, the actual performance of different Bayesian network classifiers (except the naïve Bayesian classifier) do not differ much in practice. If we assume that the data set does faithfully represent a mixture distribution, it can be concluded that the ability to capture context-specific independence, or in general, to make a better approximation to the underlying distribution, may not be critical to the performance of the classifier. On the contrary, we conjecture that there are other factors, mostly unrelated to the intricacy of the distribution, which decide the actual performance. These factors include, for example, the distribution within the testing sets, the size of the training set, and the noise present.

In general, we note that it is difficult to analyze the performance of a classifier. For instance, it has been a mystery to understand why the naïve Bayesian classifier has a robust performance even with its simplistic assumption. Often, since the training set is of a limited size, to learn a classifier with a theoretically better (likely to be more complicated) structure may not result in a classifier that performs better. Moreover, since the testing set is also small, the performance estimation is liable to random perturbations and may not reflect the true error rate. Given these two practical difficulties, the classification problem becomes complicated. Although we can stress on the *actual* performance of a classifier by evaluating on a number of real world data sets, it turns out that definition of a "good" and "bad" classifier is poorly defined. For example, if our classifier outperforms an opponent on 11 out of the 20 data sets, is it justified to claim that our classifier to be a better classifier? If

we include, in addition, three other data sets that the opponent will perform better. Can we say that our classifier is inferior?

Although it is difficult to make a reasonable conclusion, there are some general directions for future improvements. One possibility is to combine multiple classifiers, learned differently, into a single classifier. For instance, we can build a wrapper classifier such that we choose the best classifier out of a number of candidates as the representative. In another case, we can allow each classifier votes in making the decision and that the outcome is determined by the majority. In a similar vein, we can use Bayesian technique, as discussed in [12], in making the final decision. Since our learning algorithms are stochastic in nature, such strategy is particularly useful as we can learn a number of different classifiers for different initialization settings. Moreover, this reduce the risk of using a single, probably local optimal, model as the ultimate classifier.

Another possibility for improvement is to use a better parameter estimation method, like the smoothing technique in TAN. As evident in [13], the use of a better parameter estimation technique helps to improve the predictive accuracy of the naïve Bayesian network classifier on a number of data sets. In the case for learning the parameters of ABN or MN, the need is more eager because the data insufficiency problem is more severe. Often, we could not evaluate an entry in the CPT because that the particular instantiation (a particular state of the parent variables together with a particular child variable state) is not observed in the training set. If we are able to make a reasonable approximation out of the limited data, the performance of the classifier could probably be improved.

Another possible direction is to assume the presence of a hidden variable. In the works of [51, 52], it is shown that the introduction of a hidden variable could help to improve the performance of a model. In certain sense, it would imply that the restriction or the limit of the existing Bayesian network classifiers is a problem of the model itself (i.e. a network consisting only of the attribute nodes and the class node), and the hidden variable model is probably a new direction worth investigating. Among the current works, researchers have only tried introducing a hidden variable on a naïve Bayesian network classifier [52], a TAN like classifier and a multi-tree classifier [51]. Hence, it will be interesting to see the combination with

a more complex model, like ABN and MN. Nevertheless, the learning algorithm will need to be revised accordingly.

5.7 Application in Direct Marketing

In this section, we investigate the feasibility of applying Bayesian network classifiers on a real-world, data mining problem. The problem relates with direct marketing in which the objective is to predict buyers from a list of customers. Advertising campaign, which includes mailing of catalogs or brochure, is then targeted on the most promising prospects. Hence, if the prediction is accurate, it can help to enhance the *response rate* of the advertising campaign and increase the return of investment (ROI). Principally speaking, this direct marketing problem is similar to the classification problem. But instead of making a clear cut between the ones who will respond to the ones will not, the direct marketing problem requires ranking the customer list by the likelihood of purchase [7,77]. Given that Bayesian network classifiers (or Bayesian classifiers, in general) estimates the posterior probability of an instance (a customer) belonging to a particular class (active or inactive respondents), they are particularly suitable for handling the direct marketing problem.

5.7.1 The Direct Marketing Problem

Direct marketing concerns communication with prospects, so as to elicit response from them. In contrast to the mass marketing approach, direct marketing is targeted on a group of individuals that are potential buyers and are likely to give a response. In retrospect, direct marketing emerged because of the prevalence of mail ordering in the nineteenth century [58]. As technology advances, marketing is no longer restricted to mailing but includes a variety of media. Nevertheless, the most important issue in the business remains to maximize the profitability, or ROI, of a marketing campaign.

In a typical scenario, we often have a huge list of customers. This list could be records of existing customers or data bought from *list brokers*. But among the huge list, there is usually few real buyers which amounts to a few percents [11]. Since

the budget of a campaign is limited, it is important to focus the effort on the most promising prospects so that the response rate could be improved.

Before computers became widely in use, direct marketers often used simple heuristics to enhance the response rate. One straightforward approach is to use common sense to make the decision. In particular, we could match prospects by examining the demographics of the customer list. For example, in the life insurance industry, it is natural to target the advertising to ones that are rich and are aging [58]. Another common approach to enhance the response rate is to conduct list testing by evaluating the response of samplings from the list [58]. If certain group of customers give a high response rate, the actual campaign may be targeted on the customers alike. A more systematic approach, which is developed in 1920s but is still used today, is to differentiate potential buyers from non-buyers using the recency-frequency-monetary model (RFM) [58]. In essence, the profitability of a customer is estimated by three factors including the recency of buying, frequency of buying, and the amount of money one spent. Hence, only individuals that are profitable will be the targets of the campaign.

With the advancement of computing and database technology, people seek for computational approaches to assist in decision making. From the data set that contains demographic details of customers, the objective is to develop a *response model* and used the model to predict promising prospects. In certain sense, response models are similar to classifiers in the classification problem. However, unlike the classifier which makes a dichotomous decision (i.e. active or inactive respondents), the response model needs to score each customer in the data set with the likelihood of purchase. The customers are then ranked according to the score. The reason that a ranked list is desired is because it allows decision makers to select the portion of customer list to roll out to [77]. For instance, out of the 200,000 customers on the list, we might wish to sent out catalogs or brochures to the most promising 20% of customers so that the advertising campaign is cost-effective [7]. Hence, one way to evaluate the response model is to look at its performance at different *depth-of-file*. In the literature, there are various approaches proposed for building the response model. Here, we give a brief review in the section that follows.

5.7.2 Response Models

Apart from the RFM model, which could be categorized as a computational approach for direct marketing, there are other approaches that use statistical analysis, machine learning, and artificial intelligence in response model construction.

Earlier attempts often adopt a statistical analysis approach. Back in 1967, a company has already used multiple regression analysis to build the response model [58]. In 1968, the Automatic Interaction Detection (AID) system is developed which essentially uses tree analysis to divide consumers into different segments [58]. Later, the system is modified and become the Chi-Squared Automatic Interaction Detector (CHAID). One statistical analysis technique, which is still widely used today, is logistic regression. Essentially, the logistic regression model assumes that the *logit* (i.e. the logarithm of the *odd ratios*[3]) of the dependent variable (active or inactive respondents) is a linear function of the independent variables (i.e. the attributes). Because the approach is popular, newly proposed models are often compared with the logistic regression model as the baseline comparison [7,8,76].

Zahavi and Levin [76] examine the possibility of learning a back-propagation neural network as the response model. However, due to a number of practical issues and that the empirical result did not improve over a logistic regression model, it seems that the neural network approach does not bring much benefit.

Because there are striking similarities between classification and the direct marketing problem, it is straightforward to apply classification algorithms to tackle the problem. As an example, Ling and Li [49] use a combination of two well-known classifiers, the naïve Bayesian classifier and C4.5, to construct the response model. Because scoring is necessary, they modify the C4.5 classifier so that a prediction (i.e. active and inactive respondents) comes with a *certainty factor*. To combine the two classifiers, they apply ada-boosting [49] to both classifiers in learning. When they evaluate their response model across three different real-life data sets, the result shows that their approach are effective for solving the problem.

Bhattacharyya formulates the direct marketing problem as a multi-objective optimization problem [7,8]. He notes that the use of a single evaluation criterion, which

[3]The odd ratio is the ratio of the probabilities of the event happening to not happening.

is to measure the model's accuracy, is often inadequate [8]. For practical concern, he suggests that the evaluation criterion needs to include the performance of the model at a given depth-of-file. In an early attempt, he proposes to use genetic algorithms to learn the weights of a linear response model while the evaluation function is a weighted average of the two evaluation criteria. When comparing the learnt model with the logit model on a real-life data set, the new approach indicates a superior performance [7]. Recently, he attempts to use genetic programming to learn a tree-structured symbolic rule form as the response model [8]. But instead of using a weighted average criterion function, his new approach searches for *Pareto-optimal* solutions.[4] From the analysis, he finds that the GP approach outperforms the GA approach and is effective at obtaining solutions with different levels of trade-offs [8].

5.7.3 Experiment

Because Bayesian network classifiers (or Bayesian classifiers, in general) estimates the probability of belonging to certain class(es), they are also suitable to handle the direct marketing problem. By assuming the estimated probability evaluates the likelihood of purchase, a Bayesian network classifier is readily applicable to the direct marketing problem. A previous work by Ling and Li [49] has already used the idea by applying the naïve Bayesian classifier in constructing their response model. In a similar vein, it is interesting to evaluate the empirical performance of a Bayesian network response model. Specifically, we will compare the performance of the augmented Bayesian network classifier (ABN) with the traditional logistic regression approach.

Experimental Methodology

In our study, we evaluate our response models on a real-life data set. The data set contains records of customers of a specialty catalog company which mails catalog to good customers on a regular basis.[5] There is a total of 106,284 customers in the data set and each entry is described by 278 attributes.

[4] Pareto-optimal solutions are solutions that evaluates better in at least one objective function comparing to other solutions. Pareto-optimal solutions are also called non-dominated solutions.

[5] The data set is used with the permission of Dr. Man-Leung Wong's colleague.

For the train-and-test study, we split the data set into two parts, one for training the response model and one for testing. Because there are few positive examples (i.e. active respondents) in compared with negative examples, there is an imbalanced distribution. Because most learning algorithms on such type of data sets can perform badly [49], special treatment is required in creating the training set. For instance, with the number of positive examples fixed, we could mix the training set with negative examples in proportion. Here, we follows a similar approach. For the 5,740 positive examples in the data set, we partition them into two halves for the training and the testing data set. Moreover, we add twofold of negative examples to the testing set, while the rest of negative examples are assigned to the testing set. As a result, the training set has a size of 8,610 ($= 2870 \times 3$) and the testing set has a size of 97,674.

Typical in any data mining process, it is necessary to reduce the dimension of the data set by feature selection or variable transformation. For the feature selection process, there are many options possible. For instance, we could use either a *wrapper* selection process or a *filter* selection process [65]. In a wrapper selection process, different combinations are iteratively tried and evaluated by building the actual model out of the selected attributes. In a filter selection process, certain evaluation function, which is based on information theory or statistics, is defined to score a particular combination of attributes. Then, the final combination is obtained in a search process. In our current implementation, we choose to use manual selection for simplicity. In particular, we have selected nine attributes, that we regard relevant to prediction, out of the 278 attributes. Although this approach may not guarantee an optimal selection for our response models, the comparison, at least, will be fair as the models are constructed using the same attributes.

A common practice to compare the performance of different response models is to use decile analysis which estimates the enhancement of the response rate for marketing at different depth-of-file. Essentially, the scored and ranked list is equally divided into ten deciles. As a convention, customers in the zeroth decile are the top ranked customers and are predicted to give response most likely. On the other hand, customers in the ninth decile are ranked lowest. Then, a *gains table* and a *lift chart* are constructed to describe the performance of the response model. In a gains table,

various statistics at each decile are tabulated, including [64]:

Predicted Probability of Active The average of the predicted probabilities of active respondents in the decile by the response model.

Percentage of Active The actual percentage of active respondents in the decile. For a good model, the predicted probability should approximate the percentage of active respondents.

Percentage of Total Actives This calculates the ratio of the number of active respondents in the decile to the number of all active respondents.

Cumulative Percentage of Actives It is the percentage of cumulative active respondents (from decile 0 and up) over the total number of records.

Lift Lift is calculated by dividing the percentage of active respondents by the percentage of total active respondents in the file. Intuitively, it estimates the enhancement by the response model in discriminating active respondents over a random approach for the current decile.

Cumulative Lift The cumulative lift is calculated by dividing the cumulative percentage of active respondents by the percentage of total active respondents. Intuitively, this evaluates how good the response model is for a given depth-of-file over a random approach. The measure provides an important estimate of the performance of the model.

In a lift chart (or a gain chart), we depict the performance of the response model by plotting the cumulative percentage of total actives in the ten decile. A diagonal line on the chart shows the performance of a random approach.

We reuse the implementation of ABN in $\mathcal{MLC}$++in our experiment so that cross-validation and discretization are performed automatically. Besides, we also implement the logistic regression algorithm in $\mathcal{MLC}$++. Note that for logistic regression, continuous-valued attributes are not discretized. Because the system is primarily designed for classification problem, some post-processing on the system output is required to produce the decile analysis result.

The Train-and-test Result

The decile analysis of the response models of logistic regression and ABN are shown respectively in Table 5.4 and 5.5. From the tables, we can find the ABN model can discriminate more active respondents than logistic regression in the first few decile (comparing "Cum. Actives"). Hence, the lifts by ABN is higher. Probably because the training set is specially prepared, both models give an overly optimistic estimation of active respondents (comparing to the actual percentage in the decile). Although the predictions by both models are far from accurate, we can observe that ABN gives a slightly better estimate than logistic regression.

Decile	Records	Prob. of Active	Percent Active	Cum. Percent Active	Actives	% of Total Actives	Cum. Actives	Cum. % of Tot. Actives	Lift	Cum. Lift
0	9768	57.27%	10.30%	10.30%	1006	35.05%	1006	35.05%	350	350
1	9767	50.25%	4.93%	7.62%	482	16.79%	1488	51.85%	167	259
2	9768	46.54%	4.39%	6.54%	429	14.95%	1917	66.79%	149	222
3	9767	42.64%	2.50%	5.53%	244	8.50%	2161	75.30%	85	188
4	9768	38.07%	1.98%	4.82%	193	6.72%	2354	82.02%	67	164
5	9767	32.34%	1.55%	4.27%	151	5.26%	2505	87.28%	52	145
6	9767	25.83%	1.26%	3.84%	123	4.29%	2628	91.57%	42	130
7	9768	19.26%	0.94%	3.48%	92	3.21%	2720	94.77%	32	118
8	9767	13.59%	0.84%	3.19%	82	2.86%	2802	97.63%	28	108
9	9767	8.14%	0.70%	2.94%	68	2.37%	2870	100.00%	23	100
Total	97,674	33.39%	2.94%		2870	100.0%				

Table 5.4: Gains table of the result from logistic regression.

Decile	Records	Prob. of Active	Percent Active	Cum. Percent Active	Actives	% of Total Actives	Cum. Actives	Cum. % of Tot. Actives	Lift	Cum. Lift
0	9768	98.39%	11.65%	11.65%	1138	39.65%	1138	39.65%	396	396
1	9767	62.44%	5.44%	8.54%	531	18.50%	1669	58.15%	185	290
2	9768	38.33%	3.71%	6.93%	362	12.61%	2031	70.77%	126	235
3	9767	29.16%	1.74%	5.63%	170	5.92%	2201	76.69%	59	191
4	9768	21.53%	1.96%	4.90%	191	6.66%	2392	83.34%	66	166
5	9767	14.80%	1.27%	4.29%	124	4.32%	2516	87.67%	43	146
6	9767	10.09%	1.26%	3.86%	123	4.29%	2639	91.95%	42	131
7	9768	7.23%	0.92%	3.49%	90	3.14%	2729	95.09%	31	118
8	9767	4.72%	0.76%	3.19%	74	2.58%	2803	97.67%	25	108
9	9767	2.02%	0.69%	2.94%	67	2.33%	2870	100.00%	23	100
Total	97,674	28.87%	2.94%		2870	100.0%				

Table 5.5: Gains table of the result from ABN.

In Figure 5.10, the two models are compared in a gains chart. Recalling that the diagonal line represents marketing towards a random population, the two models

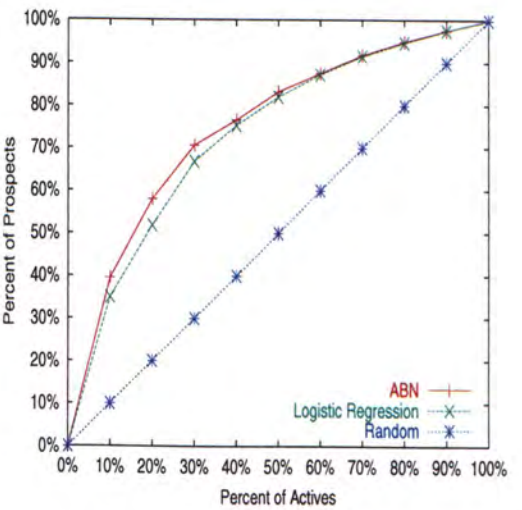


Figure 5.10: Model comparison gains chart.

clearly outperforms the random approach by discerning a larger percentage of active respondents for different depth-of-file. When comparing the two models, we can observe that ABN model performs better than logistic regression in the first few deciles (10%–40%). At later deciles, however, the models seem to deliver similar performance.

Cross-Validation Result

To make a further comparison concerning the robustness of the response models, we employ a cross-validation approach for performance estimation. Specifically, 10-fold cross-validation is used where the entire data set is partitioned at random (i.e. no catering for imbalance distribution). In Table 5.6 and 5.7, the experimental results for the logistic regression and ABN are shown respectively. Similar to the gains table, we collect the statistics on the predicted probabilities, percentage of active response, lift and cumulative lift at each decile averaged over the ten runs. The numbers beside the "$\pm$" sign are the standard deviations. From the tables, we can confirm that our earlier observations are consistent with a resampling analysis. For instance, the ABN model is observed to predict more accurately the percentage of active respondents in a decile. Moreover, it provides a better cumulative lift in the first few deciles (decile 0 to decile 4).

Decile	Predicted Prob.	Percent Actives	Lift	Cum. Lift
0	24.89% ± 0.20%	18.49% ± 1.31%	342.0 ± 17.1	342.0 ± 17.1
1	15.59% ± 0.16%	8.64% ± 0.89%	159.4 ± 15.1	250.8 ± 7.8
2	12.67% ± 0.14%	7.14% ± 0.73%	131.8 ± 11.8	211.1 ± 5.7
3	10.45% ± 0.11%	6.12% ± 0.52%	113.1 ± 10.0.	186.7 ± 4.6
4	8.50% ± 0.11%	3.74% ± 0.84%	68.7 ± 14.2	163.1 ± 2.3
5	6.63% ± 0.11%	3.02% ± 0.72%	55.0 ± 10.3	145.0 ± 1.8
6	4.89% ± 0.09%	2.38% ± 0.53%	43.3 ± 8.3	130.5 ± 1.8
7	3.45% ± 0.06%	1.82% ± 0.52%	33.3 ± 9.8	118.4 ± 1.2
8	2.36% ± 0.04%	1.50% ± 0.23%	27.5 ± 4.4	108.2 ± 0.9
9	1.38% ± 0.02%	1.14% ± 0.39%	20.6 ± 7.4	100.0 ± 0.0
Total	9.08% ± 0.08%	5.40% ± 0.31		

Table 5.6: Result of the logistic regression model.

Decile	Predicted Prob.	Percent Actives	Lift	Cum. Lift
0	27.89% ± 0.53%	21.43% ± 0.73%	397.2 ± 19.0	397.2 ± 19.0
1	10.58% ± 0.29%	8.97% ± 1.02%	165.5 ± 16.0	281.4 ± 8.9
2	6.19% ± 0.17%	5.84% ± 0.96%	107.5 ± 14.1	223.4 ± 7.2
3	4.52% ± 0.13%	5.00% ± 0.93%	92.0 ± 15.6	190.5 ± 2.8
4	3.44% ± 0.09%	3.30% ± 0.46%	60.5 ± 6.7	164.5 ± 1.6
5	2.47% ± 0.05%	2.98% ± 0.46%	54.5 ± 6.5	146.3 ± 1.0
6	1.76% ± 0.04%	2.08% ± 0.45%	37.7 ± 7.4	130.8 ± 1.0
7	1.26% ± 0.01%	1.61% ± 0.21%	29.4 ± 3.6	118.1 ± 1.0
8	0.85% ± 0.02%	1.30% ± 0.34%	23.5 ± 6.4	107.5 ± 0.7
9	0.34% ± 0.02%	1.49% ± 0.26%	27.1 ± 5.0	100.0 ± 0.0
Total	5.93% ± 0.10%	5.40% ± 0.31		

Table 5.7: Result of the ABN model.

In Figure 5.11, we depict the comparison of the cumulative lifts of the two models. Besides the averaged measure, the 95% confidence interval estimate[6], appearing as error-bar, is also shown.

Considering that an advertising campaign often involves huge investment, a response model which can categorize more prospects into the target list, which amounts to a small number of customers on the list, is valuable as it will enhance the response rate. From the experimental result, it seems that the ABN model is more desirable than the traditional logistic regression approach. Although the models are tested on a single data set (and for our particular choice of variables), it suggests a viable alternative for response model construction using Bayesian networks.

[6]We assume that the cumulative lift follows a normal distribution.

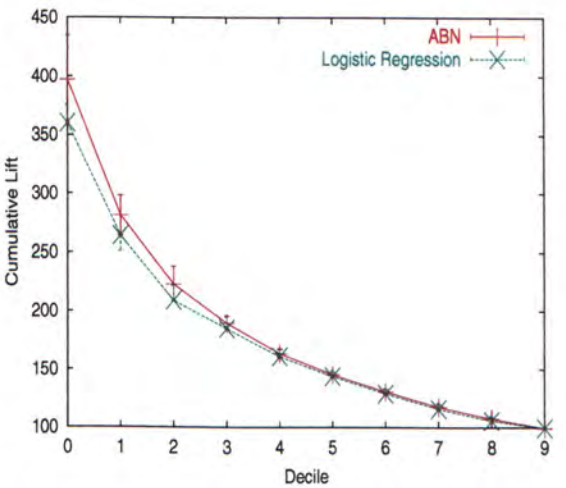


Figure 5.11: Comparing the cumulative lifts of the two models.

5.8 Summary

In this chapter, we apply our Bayesian network learning algorithm on constructing Bayesian network classifiers. Because the problem is not straightforward, we adopt two approaches that can make use of our learning algorithm: ABN and MN. Although Friedman et al. have also evaluated the two classifiers, it seems that they use heuristic search for classifier construction. Thus, it is possible that the evaluation is unfair as the resultant model may be inferior. In contrast, we attempt to use our evolutionary computation for learning the classifiers, which may provide a better study.

On evaluating the performance of our proposed classifiers, we make a formal and fair comparison by implementing atop of $\mathcal{MLC}$++. In comparing ABN and MN with the state-of-the-art classifier, TAN, we find that our classifier can compare favorably with TAN. On the other hand, our classifiers evidently outperforms the naïve Bayesian network classifier. Although ABN and MN do not provide a particular performance boost, we note the domains that they excel and discuss a few possible alternatives for improvement.

When applying a Bayesian network classifier on the direct marketing problem, we find that the result is promising. Although the result cannot be readily generalized, it possibly suggests a future direction worth investigating.

Chapter 6

Conclusion

We conclude our works with a summary of our contributions and discuss some possible future research directions.

6.1 Summary

At the beginning of this thesis, we state that learning Bayesian networks from data is a difficult problem. Although there are two different approaches to tackle the problem, they both suffer from some general drawbacks. Some recent works use evolutionary computation to handle the problem, which is a plausible approach as it is a powerful search methodology. However, when studying a learning algorithm which uses evolutionary programming, we find that the algorithm often executes slowly in practice.

Targeting on the problem, we propose two strategies that help to improve the efficiency. The first one is a hybrid learning framework, which is a combination of the dependency analysis and the score-and-search approach. It consists of two phases: in the first phase, we conduct conditional independency tests; in the second phase, we perform network searching exploiting the previous test results. Essentially, the result of testing helps to reduce the search space of the network searching problem that follows. Hence, it is expected that the search problem is made easier and there will be a gain in efficiency. Note that the proposed framework is a general methodology in which various dependency analysis methods and search methods

can be readily integrated.

The second strategy is a recombination strategy, which is helpful in evolutionary searching. As we note that "exploration and exploitation" is the key issue in evolutionary computation, an effective, and application-specific, genetic operator is very useful. Our proposed network operator, called merge, recombines two given network to produce a better offspring network. Although the operator is primarily a heuristic formulation, we find that it is effective in practice.

Based on the two new strategies, we develop two evolutionary computation schemes. Our first scheme employs a modular decomposition approach to tackle the search problem, which is inspired by the decomposability of the evaluation function. In the literature, we find that cooperative coevolution shares a similar spirit of problem decomposition. However, because there are problems in the original framework, we modify it and find that the new framework has satisfactory performance. Our second scheme is a natural extension of the evolutionary programming approach in which we incorporate the hybrid framework together with the merge operator. In particular, we circumvent the requirement of selecting a proper cutoff value (used in the hybrid framework) by adopting a different cutoff value in each candidate solution. In the experiment, we find that the new learning algorithms improve over the previous approach, which is as expected.

To apply our work on real world problems, we extend our proposed learning algorithm in learning Bayesian network classifiers. Because previous study did not target on learning an optimal classifier, we conjecture that the existing greedy heuristic may result in inferior models. Thus, the evaluation may be unfair. In our work, we apply our proposed algorithm for learning the classifiers, which will be a better approach for assessing the performance of the classifiers. In comparing with some well-known classifiers, we find that the performance of our classifier largely equals the rivals. Based on the observation, we give a general discussion and mention about possible improvements. Finally, we also provide a study of applying Bayesian network classifiers on the direct marketing problem. Because the results are promising, it suggests a worth investigating direction for future research.

6.2 Future Work

Because our learning algorithms are efficient, we can apply them on large-scale problems where there are many attributes and the size of the data set is huge. Also, since conditional independence test result will be more reliable for larger data set, our algorithms favors for solving large-scale problem. However, when we handle large data set, an obvious problem will be that the time required on evaluating the MDL metric will also scale up. Currently, we use a simple and direct approach for metric evaluation. As the size of the data set grows, the time spent on a single evaluation will also grow. Hence, it is important to optimize the metric evaluation operation so that the algorithm is fast even for a large-scale problem. This could be achieved by using a better database management system, an optimized data structure or even a simpler evaluation method that approximates the true score. From another perspective, the learning algorithm can be more efficient if it adapts to, for example, delicate query ordering [27], which optimize the metric evaluation operation.

Throughout our work, we have assumed that the learning data is complete. However, for most real world problems, our assumption can be violated: many data entries are missing and there may be hidden variables governing the observations. Obviously, the learning problem will be more difficult and it will not be trivial to define or to evaluate the scoring metric. A noteworthy work in this category is the structural EM algorithm [25] which tackles the problem by incorporating the EM algorithm. Recently, Myers et al. [53] attempts to use evolutionary algorithms to handle the problem [53]. Possibly, we can follow the direction and develop efficient approach for learning Bayesian networks from incomplete data. Furthermore, we can also investigate the unsupervised learning problem [71] which should be closely related to our studied problem.

We can explore new alternatives that Bayesian networks are applied on real world problems. Although a Bayesian networks is useful on its own, it is possible to extend the basic prototype to suit particular needs. For instance, there are previous works that combines Bayesian networks with decision theory [14], extends the single network into a multinet model [29], and transforms a network to describe

temporal information (i.e. dynamic Bayesian network) [42]. Hence, the potential use is unlimited as long as we *know* how to use it.

Appendix A

A Supplementary Parameter Study

In Section 4.3 and 4.4, we study the effect of different parameter settings on our proposed algorithms CCGA and HEP respectively. However, since the experiments are conducted on a single data set, ALARM-O, the result may vary on other data sets. In this chapter, we conduct a similar study on another data set, PRINTD-5000. We argue that the observed trends or variations that we present in the previous sections should be valid in general. However, the degree of influence of various parameter settings may vary across different data sets. Here, we summarize the result in a concise form so that a comparison could be made easily.

A.1 Study on CCGA

A.1.1 Effect of Different α

From Table A.1, we find that:

- The case for $\alpha = 1.0$ gives poor result as it cannot find the optimal solution.
- For other cases, CCGA is able to obtain the optimal solution. However, the time used and metric evaluations increase with larger α value.
- Moreover, it would require longer generations to obtain the best-so-far solution

	AFS	AIS	AET	ANG	AME	ASD
$\alpha = 0.02$	106,541.6 (0.0)	110,189.2 (618.4)	35.4 (0.3)	14.6 (9.0)	591.8 (14.9)	0.0 (0.0)
$\alpha = 0.05$	106,541.6 (0.0)	110,189.2 (618.4)	35.8 (0.3)	14.6 (9.0)	591.8 (14.9)	0.0 (0.0)
$\alpha = 0.3$	106,541.6 (0.0)	114,622.2 (1,132.8)	64.7 (1.4)	12.7 (7.0)	2,540.6 (56.4)	0.0 (0.0)
$\alpha = 0.5$	106,541.6 (0.0)	115,051.1 (1,250.5)	107.0 (3.2)	22.7 (19.8)	7,266.7 (92.3)	0.0 (0.0)
$\alpha = 1.0$	106,681.8 (66.9)	114,757.9 (1,157.4)	1,722.0 (12.6)	757.1 (172.8)	256,068.7 (1,914.9)	6.5 (3.4)

Table A.1: Performance of CCGA with different α.

	AFS	AIS	AET	ANG	AME	ASD
$m = 20$	106,541.6 (0.0)	114,622.2 (1,132.8)	64.7 (1.4)	12.7 (7.0)	2,540.6 (56.4)	0.0 (0.0)
$m = 30$	106,541.6 (0.0)	114,741.1 (1,060.0)	67.4 (2.6)	9.6 (4.7)	2,586.2 (43.3)	0.0 (0.0)
$m = 40$	106,541.6 (0.0)	114,602.1 (933.9)	68.2 (2.8)	9.2 (4.6)	2,622.9 (45.2)	0.0 (0.0)
$m = 50$	106,541.6 (0.0)	114,475.7 (918.0)	70.1 (1.3)	10.1 (4.4)	2,674.8 (47.7)	0.0 (0.0)

Table A.2: Performance of CCGA under different population sizes.

for a larger α value.

Thus, provided that a smaller value would not bring undesirable side-effect, using a smaller α is recommended.

A.1.2 Effect of Different Population Sizes

From Table A.2, we find that:

- Time used and metric evaluations increase with a larger population size.
- The expected benefit of using a larger population is not obvious. Indeed, the difference in ANG of the four different runs are not significant.

Thus, a smaller population size seems to suffice.

A.1.3 Effect of Varying Crossover and Mutation Probabilities

From Table A.3, we find that:

	AFS	AET	ANG	AME	ASD
$p_c = 0.5$	106,541.6 (0.0)	60.7 (0.9)	11.2 (6.8)	1,848.9 (34.9)	0.0 (0.0)
$p_c = 0.7$	106,541.6 (0.0)	59.6 (2.5)	11.5 (6.6)	1,855.6 (46.2)	0.0 (0.0)
$p_c = 0.9$	106,541.6 (0.0)	57.7 (2.0)	13.3 (7.1)	1,844.5 (38.4)	0.0 (0.0)

(a) $p_m = 0.05$

	AFS	AET	ANG	AME	ASD
$p_c = 0.5$	106,541.6 (0.0)	65.8 (2.7)	9.8 (4.8)	2,549.4 (46.9)	0.0 (0.0)
$p_c = 0.7$	106,541.6 (0.0)	64.7 (1.4)	12.7 (7.0)	2,540.6 (56.4)	0.0 (0.0)
$p_c = 0.9$	106,541.6 (0.0)	63.9 (1.4)	12.6 (6.3)	2,521.0 (46.7)	0.0 (0.0)

(b) $p_m = 0.2$

	AFS	AET	ANG	AME	ASD
$p_c = 0.5$	106,541.6 (0.0)	72.1 (5.7)	10.6 (4.5)	3,244.7 (50.5)	0.0 (0.0)
$p_c = 0.7$	106,541.6 (0.0)	70.7 (3.7)	11.4 (4.1)	3,229.7 (50.6)	0.0 (0.0)
$p_c = 0.9$	106,541.6 (0.0)	74.5 (7.3)	11.7 (5.1)	3,192.5 (60.4)	0.0 (0.0)

(c) $p_m = 0.5$

Table A.3: Performance of CCGA with different p_c and p_m.

- The variations in AET and AME induced by the change by p_m is greater than that by p_c.
- A larger crossover probability seems to cause AME to drop.
- Although the difference is not significant, it seems that a smaller crossover probability can help to obtain the final solution faster.

Thus, our recommendation that we should use a low mutation probability and a not-too-high crossover probability is also justified for this data set.

A.1.4 Effect of Varying Belief Factor

From Table A.4, we find that:

- In the extreme case of the belief factor equals 1.0, CCGA get stuck at local optimal solutions.

	AFS	AET	ANG	AME	ASD
factor = 0.0	106,541.6 (0.0)	78.3 (7.6)	8.5 (2.6)	2,502.2 (51.2)	0.0 (0.0)
factor = 0.1	106,541.6 (0.0)	71.9 (6.2)	9.8 (4.6)	2,572.9 (53.7)	0.0 (0.0)
factor = 0.2	106,541.6 (0.0)	65.1 (1.3)	12.7 (7.0)	2,540.6 (56.4)	0.0 (0.0)
factor = 0.5	106,541.6 (0.0)	57.4 (0.5)	22.4 (12.8)	2,275.6 (45.1)	0.0 (0.0)
factor = 0.7	106,541.6 (0.0)	51.3 (0.5)	43.1 (38.9)	1,980.4 (40.6)	0.0 (0.0)
factor = 1.0	108,557.4 (657.2)	28.4 (1.3)	5.8 (2.5)	669.4 (96.7)	6.5 (2.3)

Table A.4: Performance comparison of different belief factor.

- A smaller belief factor can help to obtain solution in an earlier generation but with the cost of more computational time.
- The tradeoff is more clearly observed in this experiment.

Because a larger belief factor may deteriorate the performance, the recommendation of a moderate value (e.g. 0.2) is justified.

A.2 Study on HEP

A.2.1 The Hybrid Framework and the Merge Operator

Because the prevention of cycle formation is demonstrated to make a big difference on the computation time, we will not investigate the issue further. However, we still want to compare the contribution of different improvement strategies.

	AFS	AIS	AET	ANG	AME	ASD
EP+merge	106,542.0 (0.0)	116,275.9 (480.3)	221.0 (1.8)	213.5 (600.2)	15,707.8 (325.8)	0.0 (0.0)
EP+CI	106,542.0 (0.0)	111,401.3 (392.5)	134.1 (2.9)	170.6 (154.2)	1,659.1 (315.9)	0.0 (0.0)
HEP	106,542.0 (0.0)	111,401.3 (392.5)	165.3 (3.2)	43.1 (14.6)	1,754.2 (439.0)	0.0 (0.0)

Table A.5: Performance comparison of different implementations.

From Table A.5, we find that:

- HEP is not the slowest, but "EP+merge" is. It possibly suggests that the drawback of "EP+merge", that it requires more metric evaluations, surface

in this experiment. Looking at AME, our observation is further supported by that "EP+merge"requires nearly nine times more evaluations on average comparing to either "EP+CI" or "HEP".

- It is evident that HEP can obtain the final solution earliest.

Thus, it seems that "HEP" has the anticipated advantage.

A.2.2 Effect of Different Population Sizes

	AFS	AIS	AET	ANG	AME	ASD
$m = 40$	106,542.0 (0.0)	111,505.1 (389.1)	139.7 (1.9)	42.1 (13.2)	1,603.1 (283.0)	0.0 (0.0)
$m = 50$	106,542.0 (0.0)	111,401.3 (392.5)	165.3 (3.2)	43.1 (14.6)	1,754.2 (439.0)	0.0 (0.0)
$m = 60$	106,542.0 (0.0)	111,286.1 (387.0)	190.0 (2.6)	37.8 (10.3)	1,824.5 (325.8)	0.0 (0.0)

Table A.6: Performance of HEP with different population sizes.

From Table A.6, we find that:

- A larger population also implies increases in the time used and metric evaluations.

- However, the advantage of using a larger population seems to manifest as it seems to help to obtain the final solution in a smaller number of generations.

Because there is a tradeoff (execution time v.s. generation) involved in using a larger population, a moderate population size is recommended.

A.2.3 Effect of Different Δ_α

From Table A.7, we find that:

- The previously observed trend on AET and AME is not obvious in the current experiment.

- However, our choice is shown to deliver satisfactory performance.

	AFS	AET	ANG	AME	ASD
$\Delta_\alpha = 0.0$	106,542.0 (0.0)	165.7 (3.5)	42.4 (14.6)	1,759.6 (399.5)	0.0 (0.0)
$\Delta_\alpha = 0.005$	106,542.0 (0.0)	167.3 (3.2)	42.0 (12.3)	1,793.7 (379.1)	0.0 (0.0)
$\Delta_\alpha = 0.01$	106,542.0 (0.0)	166.1 (4.1)	41.3 (10.8)	1,839.5 (535.9)	0.0 (0.0)
$\Delta_\alpha = 0.02$	106,542.0 (0.0)	165.3 (3.2)	43.1 (14.6)	1,754.2 (439.0)	0.0 (0.0)
$\Delta_\alpha = 0.05$	106,542.0 (0.0)	192.6 (4.0)	37.0 (7.7)	1,810.6 (520.7)	0.0 (0.0)
$\Delta_\alpha = 0.1$	106,542.0 (0.0)	166.6 (0.8)	44.1 (12.2)	2,006.1 (137.0)	0.0 (0.0)

Table A.7: Performance of HEP with different Δ_α.

A.2.4 Efficiency of the Merge Operator

Similar to the previous study, we depict in Figure A.1 the number of successful merge operations during a run. From the collected figures, the average number equals 4.95, so that the success rate also amounts to about one-fifth (4.95/25).

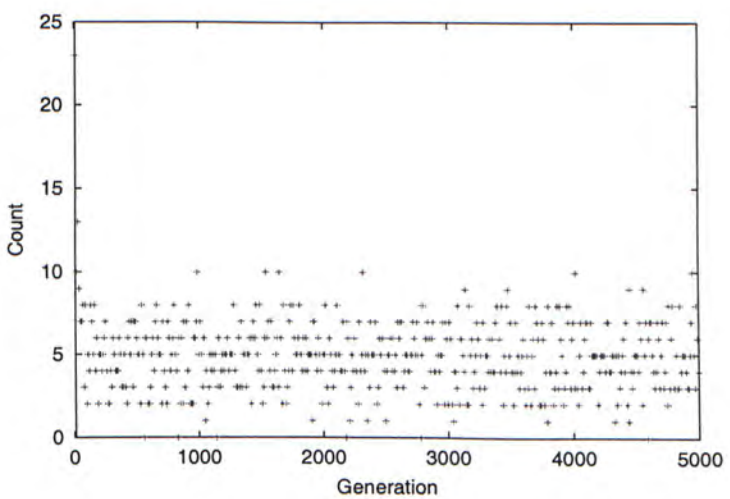


Figure A.1: Successful merging in a run (PRINTD data set).

Bibliography

[1] Acid, S., and de Campos, L. M. BENEDICT:an algorithm for learning probabilistic belief networks. In *Proceedings of the IPMU-96 Conference* (1996).

[2] Agresti, A. *Categorical Data Analysis.* Wiley, New York, circa 1990.

[3] Banzhaf, W., Nordin, P., Keller, R. E., and D.Francone, F. *Genetic Programming: An Introduction.* Morgan Kaufmann, San Francisco, California, 1998.

[4] Beasley, D., Bull, D. R., and Martin, R. R. An overview of genetic algorithms: Part 1, fundamentals. *University Computing 15*, 2 (1993), 58–69. ftp://ralph.cs.cf.ac.uk/pub/papers/GAs/ga_overview1.ps.

[5] Beaumont, G. P., and Knowles, J. D. *Statistical Tests: An introduction with MINITAB commentary.* Prentice-Hall, 1996.

[6] Beinlinch, I., Suermondt, H., Chavez, R., and Cooper, G. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of Second European Conference Artificial Intelligence in Medicine* (1989), pp. 247–256.

[7] Bhattacharyya, S. Direct marketing response models using genetic algorithms. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (1998), pp. 144–148.

[8] Bhattacharyya, S. Evolutionary algorithms in data mining: Multi-objective performance modeling for direct marketing. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining* (August 2000), pp. 465–473.

[9] Blake, C. L., and Merz, C. J. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[10] BOUCKAERT, R. R. *Bayesian Belief Networks: from Inference to Construction.* PhD thesis, Utrecht University, June 1995.

[11] CABENA, P., HADJINIAN, P., STADLER, R., VERHEES, J., AND ZANSI, A. *Discovering Data Mining: From Concept to Implementation.* Prentice-Hall Inc., 1997.

[12] CERQUIDES, J. Applying general Bayesian techniques to improve tan induction. In *International Conference on Knowledge Discovery and Data Mining* (San Diego, California, USA, August 1999), pp. 292–296.

[13] CESTNIK, B. Estimating probabilities: A crucial task in machine learning. In *The Ninth European Conference on Artificial Intelligence* (Stockholm, Sweden, August 1990), L. C. Aiello, Ed., Pitman Publishers, pp. 147–149.

[14] CHARNIAK, E. Bayesian networks without tears. *AI Magazine 12*, 4 (1991), 50–63.

[15] CHENG, J., BELL, D. A., AND LIU, W. Learning Bayesian networks from data: An efficient approach based on information theory. In *Proceedings of the Sixth ACM International Conference on Information and Knowledge Management* (Las Vegas, Nevada, November 1997), ACM, pp. 325–331.

[16] CHENG, J., AND GREINER, R. Comparing Bayesian network classifiers. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (Stockholm, Sweden, July–August 1999), K. B. Laskey and H. Prade, Eds., Morgan Kaufmann, pp. 101–108.

[17] CHICKERING, D. M., GEIGER, D., AND HECKERMAN, D. Learning Bayesian network is NP-Hard. Tech. rep., Microsoft Research, 1994.

[18] CHOW, C. K., AND LIU, C. N. Approximating discrete probability distributions with dependency trees. *IEEE Transactions on Information Theory 14* (1968), 462–467.

[19] DASH, D., AND DRUZDZEL, M. J. A hybrid anytime algorithm for the construction of causal models from sparse data. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (Stockholm, Sweden, July–August 1999), K. B. Laskey and H. Prade, Eds., Morgan Kaufmann, pp. 142–149.

[20] DE CAMPOS, L. M., AND HUETE, J. F. Approximating causal orderings for Bayesian networks using genetic algorithms and simulated annealing. Tech. rep., Department

of Computer Science and Artificial Intelligence, University of Granada, Spain, May 1999.

[21] DOMINGOS, P., AND PAZZANI, M. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning 29* (1997), 103–130.

[22] DUDA, R. O., AND HART, P. E. *Pattern Classification and Scene Analysis.* John Wiley & Sons Inc., 1973.

[23] EZAWA, K. J., AND NORTON, S. W. Constructing Bayesian networks to predict uncollectible telecommunications accounts. *IEEE Expert-Intelligent Systems and Their Applications 11* (October 1996), 45–51.

[24] FREUND, J. E. *Mathematical Statistics*, fifth ed. Prentice Hall, 1992.

[25] FRIEDMAN, N. The Bayesian structural EM algorithm. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence* (Wisconsin, USA, July 1998), G. F. Cooper and S. Moral, Eds., Morgan Kaufmann, pp. 129–138.

[26] FRIEDMAN, N., GEIGER, D., AND GOLDSZMIDT, M. Bayesian network classifiers. *Machine Learning 29* (1997), 131–163.

[27] FRIEDMAN, N., AND GETOOR, L. Efficient learning using constrained sufficient statistics. In *Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics* (Fort Lauderdale, Florida, January 1999), D. Heckerman and J. Whittaker, Eds., Morgan Kaufmann.

[28] FUNG, R. M., AND CRAWFORD, S. L. Constructor: A system for the induction of probabilistic models. In *Proceedings of the Seventh National Conference on Artificial Intelligence* (1990), pp. 762–769.

[29] GEIGER, D., AND HECKERMAN, D. Knowledge representation and inference in similarity networks and Bayesian multinets. *Artificial Intelligence 82* (1996), 45–74.

[30] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[31] HADDAWY, P. An overview of some recent developments in Bayesian problem solving techniques. Introduction to AI Magazine Special Issue on Uncertainty in Artificial Intelligence, Summer, 1999.

[32] HECKERMAN, D. A tutorial on learning Bayesian networks. Tech. rep., Microsoft Research, Advanced Technology Division, March 1995.

[33] HECKERMAN, D., AND HORVITZ, E. Inferring informational goals from free-text queries: A Bayesian approach. In *Proceedings of Fourteenth Conference of Uncertainty in Artificial Intelligence* (Wisconsin, USA, July 1998), G. F. Cooper and S. Moral, Eds., Morgan Kaufmann, pp. 230–237.

[34] HECKERMAN, D., AND WELLMAN, M. P. Bayesian networks. *Communications of the ACM 38*, 3 (March 1995), 27–30.

[35] HENRION, M. Propagating uncertainty in Bayesian networks by logic sampling. In *Proceedings of the Second Conference on Uncertainty in Artificial Intelligence* (Amsterdam, North Holland, 1988), J. Lemmar and L. Kanal, Eds., pp. 149–163.

[36] HERSKOVITS, E., AND COOPER, G. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning 9*, 4 (1992), 309–347.

[37] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[38] KEOGH, E. J., AND PAZZANI, M. J. Learning augmented Bayesian classifiers: A comparison of distribution-based and classification-based approaches. In *Proceedings of the Seventh International Workshop on AI and Statistics* (Fort Lauderdale, Florida, January 1999), D. Heckerman and J. Whittaker, Eds., Morgan Kaufmann, pp. 225–230.

[39] KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence (IJCAI)* (Montreal, Canada, August 1995), Morgan Kaufmann, pp. 1137–1145.

[40] KOHAVI, R., AND SOMMERFIELD, D. *MLC++ Utilities*, second ed., October 1996. http://www.sgi.com/tech/mlc/util/util.ps.

[41] KOHAVI, R., SOMMERFIELD, D., AND DOUGHERTY, J. Data mining using MLC++: A machine learning library in C++. In *Proceedings of the Eighth International Conference on Tools with Artificial Intelligence* (November 1996), IEEE Computer Society Press, pp. 234–245. http://www.sgi.com/tech/mlc/index.html.

[42] KOLLER, D. Course notes. http://www.stanford.edu/class/cs228/. Course CS228, "Probabilistic Models in Artificial Intelligence", at Stanford University.

[43] KONONENKO, I. Semi-naïve Bayesian classifier. In *Proceedings of the Sixth European Working Session on Learning* (Porto, Portugal, 1991), Springer-Verlag, pp. 206–219.

[44] KOZA, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[45] LAM, W., AND BACCHUS, F. Learning Bayesian belief networks-an approach based on the MDL principle. *Computational Intelligence 10*, 4 (1994), 269–293.

[46] LANGLEY, P., AND SAGE, S. Induction of selective Bayesian classifier. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence* (Seattle, Washington, July 1994), R. L. de Mantaras and D. Poole, Eds., Morgan Kaufmann.

[47] LARRAÑAGA, P., KUIJPERS, C., MURA, R., AND YURRAMENDI, Y. Structural learning of Bayesian network by genetic algorithms: A performance analysis of control parameters. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*, 9 (September 1996), 912–926.

[48] LAURITZEN, S. L., AND SPIEGLHALTER, D. J. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of Royal Statistics Society 50*, 2 (1988), 157–194.

[49] LING, C. X., AND LI, C. Data mining for direct marketing: Problems and solutions. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (1998), pp. 73–79.

[50] MANI, S., AND VALTORTA, M. Bayesian model building from a large medical dataset: A practical study. In *Working Notes of the Fifth International Workshop on Artificial Intelligence and Statistics* (Fort Lauderdale, Florida, 1995), F. Flinstone and B. Bunny, Eds., pp. 266–271.

[51] MEILĂ-PREDOVICIU, M. *Learning with Mixtures of Trees.* PhD thesis, Massachusetts Institute of Technology, January 1999.

[52] MONTI, S., AND COOPER, G. F. A Bayesian network classifier that combines a finite mixture model and a naïve-bayes model. In *Proceedings of the Fifteenth Conference*

on Uncertainty in Artificial Intelligence (Stockholm, Sweden, July–August 1999), K. B. Laskey and H. Prade, Eds., Morgan Kaufmann, pp. 447–456.

[53] MYERS, J. W., LASKEY, K. B., AND DEJONG, K. A. Learning Bayesian networks from incomplete data using evolutionary algorithms. In *Proceedings of the First Annual Conference on Genetic and Evolutionary Computation Conference 1999* (Orlando, Florida, USA, July 1999), Morgan Kauffman, pp. 458–465.

[54] NEAPOLITAN, R. *Probabilistic Reasoning in Expert Systems.* Wiley, New York, 1990.

[55] Norsys Bayes net library. http://www.norsys.com/net_library.htm.

[56] OLESEN, K. G., KJÆRULFF, U., JENSEN, F., JENSE, F. V., FALCK, B., ANDREASSEN, S., AND ANDERSEN, S. K. A MUNIN network for the median nerve - a case study in loops. *Applied Artificial Intelligence 3* (1989), 385–404.

[57] PEARL, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[58] PETRISON, L. A., BLATTBERG, R. C., AND WANG, P. Database marketing: Past present, and future. *Journal of Direct Marketing 11*, 4 (1997), 109–125.

[59] POTTER, M. A. *The Design and Analysis of a Computational Model of Cooperative Coevolution.* PhD thesis, George Mason University, 1997.

[60] POTTER, M. A., AND DE JONG, K. A. Evolving neural networks with collaborative species. In *Proceedings of the 1995 Summer Computer Simulation Conference* (Ottawa, Canada, 1995).

[61] POTTER, M. A., DEJONG, K. A., AND GREFENSTETTE, J. A coevolutionary approach to learning sequential decision rules. In *Proceedings of the Sixth International Conference (ICGA'95)* (July 1995), pp. 366–372.

[62] QUINLAN, R. C4.5 release 8. http://www.cse.unsw.edu.au/~quinlan/c4.5r8.tar.gz.

[63] RISSANEN, J. Modelling by shortest data description. *Automatica 14* (1978), 465–471.

[64] RUD, O. P. *Data Mining Cookbook: modeling data for marketing, risk and customer relationship management.* Wiley, New York, 2001.

[65] SINGH, M. *Learning Bayesian Networks for Solving Real-World Problems.* PhD thesis, University of Pennsylvania, 1998.

[66] SINGH, M., AND VALTORTA, M. An algorithm for the construction of Bayesian network structures from data. In *Proceedings of the Ninth Conference of Uncertainty in Artificial Intelligence* (San Mateo, CA, 1993), D. Heckerman and E. H. Mamdani, Eds., Morgan Kaufmann, pp. 259–265.

[67] SPATZ, C., AND JOHNSTON, J. O. *Basic statistics: tales of distribution*, second ed. Brooks/Cole Publishing Company, Monterey, California, 1981.

[68] SPIRTES, P., GLYMOUR, C., AND SCHEINES, R. Independence relations produced by parameter values in causal models. *Philosophical Topics 18*, 2 (1990), 55–70.

[69] SPIRTES, P., GLYMOUR, C., AND SCHEINES, R. *Causation, Prediction, and Search*, vol. 81 of *Lecture Notes in Statistics.* Springer-Verlag, New York, 1993.

[70] SUZUKI, J. Learning Bayesian belief networks based on the minimum description length principle: Basic properties. In *IEICE Transactions Fundamentals* (November 1999), vol. E82-A.

[71] THIESSON, B., MEEK, C., CHICKERING, D. M., AND HECKERMAN, D. Learning mixtures of Bayesian networks. Tech. Rep. MSR-TR-97-30, Microsoft Research, Redmond, Washington, February 1998.

[72] TIAN, J. A branch-and-bound algorithm for MDL learning Bayesian networks. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence* (San Francisco, California, June–July 2000), C. Boutilier and M. Goldszmidt, Eds., Morgan Kaufmann, pp. 580–588.

[73] VAN DER GAAG, L. C. Bayesian belief networks: Odds and ends. *The Computer Journal 39*, 2 (1996), 97–113.

[74] WEISS, S. M., AND KULIKOWSKI, C. A. *Computer Systems that learn: classification and Prediction Methods from Statistics, Neural Nets, Machine Learning and Expert Systems.* Morgan Kaufmann Publishers, Inc., 1991.

[75] WONG, M. L., LAM, W., AND LEUNG, K. S. Using evolutionary programming and minimum description length principle for data mining of Bayesian networks. *IEEE*

Transactions on Pattern Analysis and Machine Intelligence 21, 2 (February 1999), 174–178.

[76] ZAHAVI, J., AND LEVIN, N. Applying neural computing to target marketing. *Journal of Direct Marketing 11*, 4 (1997), 76–93.

[77] ZAHAVI, J., AND LEVIN, N. Issues and problems in applying neural computing to target marketing. *Journal of Direct Marketing 11*, 4 (1997), 63–75.